UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

ESTUDIO DE UN SISTEMA MÓVIL DE TELEVISIÓN CON SOPORTE DE PAGO POR VISIÓN A TRAVÉS DE REDES DE PRÓXIMA GENERACIÓN

Autor: D. Antonio González Pardo
Tutor: Prof. D. Daniel Díaz Sánchez

Septiembre de 2009

Agradecimientos

Me gustaría dar las gracias a mi tutor, Daniel Díaz Sánchez, por toda la paciencia que ha tenido conmigo y por atenderme cuando me presentaba en su despacho sin previo aviso. Además, sería injusto no darle las gracias a Davide Proserpio por su ayuda y dedicación en aquellas cuestiones relacionadas con el servidor.

Finalmente, debo agradecer a Jean-Baptiste Kempf, a Pierre Ynard y a Geoffroy Couprie, cuyos alias respectivamente son j-b, linkfanel y geal. Por su ayuda a través del foro de videolan y del canal de IRC #videolan, en los temas de migración del programa VLC a la plataforma Windows CE.

Resumen

Las redes de próxima generación, o NGN, es la tecnología que relevará al actual sistema de tercera generación, 3G. Estas redes están basadas en tecnología IP y proporcionan servicios de datos, telefonía y multimedia. El principal objetivo es mezclar la tecnología IP con los dispositivos móviles. Desde el punto de vista de los operadores, las redes NGN permiten la integración en la red de servicios desarrollados por terceras personas. Y además el proceso de desarrollo de un nuevo servicio es más rápido y barato. Desde el punto de vista del usuario, ofrece un amplio rango de servicios que podrán ser utilizados desde cualquier dispositivo con acceso a Internet. Este dispositivo puede ser un teléfono móvil, el ordenador de la oficina o el ordenador de su casa.

El objetivo de este documento es describir el estudio de un sistema de televisión móvil basado en redes de próxima generación. El documento expone un estado del arte sobre las tecnologías relacionadas con este proyecto y el proceso de desarrollo del proyecto. Finalmente, se presentarán los resultados del estudio y desarrollo del proyecto y se propondrán unas futuras líneas de desarrollo.

Abstract

Next Generation Networking, or NGN, is the technology that will replace the current third generation system, 3G. These networks are based on IP technology and provide data, voice and multimedia services. The aim is the merge of IP technology and the mobile devices. From the operators point of view, NGN allows to integrate in the network services developed by third parties. And also, the development process of a new service is faster and cheaper. From the users point of view, it allows a wide range of services that will be used from any device with access to Internet. This device could be a mobile phone, an office computer or a personal computer.

The aim of this document is to describe the study of a mobile television system based on Next Generation Networking. The document presents the state-of-the-art about the technologies related with this project and the development process of it. Finally, the results of the study and development of the project are explained, and the future ways of development will be proposed.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se pretende dar a conocer la motivación que ha llevado al desarrollo de este proyecto, así como los objetivos de mismo. En la última sección, se resumen los contenidos de los siguientes capítulos.

1.1. Motivación del proyecto

Para entender la motivación del proyecto es necesario analizar la situación del mercado de la telefonía móvil y el sistema de televisión actual.

Según el último informe trimestral de la Comisión del Mercado de las Comunicaciones [1], en el primer trimestre de este año el número de líneas de dispositivos móviles asciende a *50.405.751* líneas.

Si tenemos en cuenta la nota de prensa publicada por el padrón municipal el 3 de Junio del 2009 [2], la población española, a principios del 2009 era de *46,6* millones de personas.

Con estos dos datos podemos observar que por término medio cada persona posee *1,08* dispositivos móviles:

$$Media = \frac{DispositivosMoviles}{Densidad} = \frac{50{,}405{,}751}{46{,}600{,}000} = 1,08 \qquad (1.1)$$

Este dato es orientativo, ya que de los 46,6 millones de personas se deben descontar a los niños y las personas mayores que debido a su edad no disponen y/o no pueden utilizar un dispositivo móvil; por lo que la media de dispositivos móviles por persona es aún mayor. Este dato nos indica que el mercado de la telefonía móvil es un mercado saturado donde los

operadores móviles encuentran más fácil aumentar su cuota de mercado captando clientes de la competencia que creando clientes nuevos.

Para que una compañía de telefonía móvil aumente su cuota de mercado de una manera significante, debe proporcionar a sus clientes unos servicios que sean innovadores y atractivos para el mercado objetivo y, a la vez, deben de ser exclusivos, para asegurarse un incremento en el número de clientes.

En lo referente a las plataformas de televisión, en nuestro país las dos plataformas más conocidas son la televisión vía satélite y la televisión digital terrestre. Sin embargo, estas no son las únicas plataformas existentes ya que, por ejemplo, existe la televisión por cable o la televisión para dispositivos móviles.

Resulta interesante la diferencia entre la penetración de los dispositivos móviles, mencionada anteriormente, y la penetración de las plataformas de TV móviles, la cual es prácticamente nula en nuestro país. Por ello, creemos interesante el estudio de un sistema de TV móvil así como el estudio de una aplicación para el uso de dicho sistema desde un terminal móvil. Además, se intentará abordar el tema de protección de contenidos para la salvaguarda de los derechos de los proveedores de contenido y poder crear así un sistema de *Pago por visión* o *PPV*.

1.2. Objetivos

Para intentar mejorar la situación expuesta en la sección anterior y aumentar la tasa de penetración de las plataformas de televisión móvil en nuestro país, se definen los siguientes objetivos que marcarán el rumbo de este proyecto:

- Estudiar IP Multimedia Subsystem, IMS [3], como plataforma para el despliegue de servicios de televisión móvil.
- Estudiar el estado del arte de los reproductores para dispositivos limitados.
- Desarrollar un cliente de IP Television, IPTV [4], que sea capaz de gestionar la sesión multimedia solicitada por el usuario con el servidor que proporciona el servicio.
- Desarrollo de un sistema de seguridad que permita la protección del vídeo utilizando un sistema de cifrado en las capas MPEG-2 [5].

A parte de estos objetivos , se plantean otros objetivos secundarios que mejorarán la calidad de nuestro sistema. Estos objetivos secundarios afectarán al código del sistema y facilitarán cualquier ampliación del proyecto en un futuro.

A continuación se describen estos objetivos secundarios:

- El sistema debe de ser robusto y debe ser capaz de recuperarse ante posibles fallos. Entendiendo por *recuperarse* que el sistema debe de continuar activo cuando se produzca algún fallo, dándole al usuario la potestad de decidir qué hacer con el sistema: solventar el error y continuar con el uso de la aplicación o cerrarla.

- El sistema debe de ser reutilizable. Una vez desarrollado el proyecto, sería muy recomendable que se pudiese utilizar la totalidad del código, o una parte del mismo, en otros proyectos sin tener la necesidad de modificarlo en gran medida.

- El sistema debe de ser adaptable. Lo que significa que el sistema debe favorecer la adhesión de nuevos módulos o funcionalidades en un futuro, sin que ello implique un gran esfuerzo de programación.

Como lenguaje de programación se utilizará *C sharp*, C#, y la herramienta para programar será *.NET*. Este lenguaje nos permitirá portar la aplicación a todos aquellos dispositivos que estén basados en un sistema Windows y también podrán ser ejecutados en sistema UNIX bajo ciertos emuladores.

1.3. Contenido de la memoria

En esta sección de hará un breve resumen del contenido de los apartados de este documento.
El siguiente capitulo es el Estado del arte. Dicho capítulo tiene como objetivo la descripción de los conceptos más importantes relacionados con este proyecto. Para ello, la mayor parte del capítulo está dedicada a la arquitectura de una red IMS. Se describe qué es una red IMS, sus características, ventajas y requisitos. Después se analiza con detalle todos los componentes de una red IMS analizando tanto las funciones que debe de tener como las interfaces necesarias para establecer la comunicación entre las funciones. Además en

dicho capítulo se realiza un estudio sobre las principales plataformas de televisión para dispositivos móviles. Finalmente, también se analizan las implementaciones SIP que se pueden encontrar en Internet y se realiza un estudio de los reproductores que se encuentran disponible para dispositivos móviles basados en Windows CE.

Después de esto, en los capítulos 3 y 4, se describe el proceso de desarrollo del proyecto. Se ha decidido separar este proceso en dos capítulos porque el capítulo 3 contiene el proceso de desarrollo del cliente de IPTV que es el objetivo de este proyecto. Mientras que el capítulo 4 contiene el proceso de desarrollo de un reproductor para dispositivos móviles.

El capítulo 5 pretende describir la planificación llevada a cabo durante el desarrollo del proyecto, así como proponer un presupuesto estimado. Además de esto, se explican los problemas más importantes que se han producido durante el desarrollo y que han proporcionado un retraso importante en la planificación del proyecto.

Una vez desarrollado todo el proyecto, es necesario someterlo a un plan de pruebas para cerciorarse del correcto funcionamiento de la aplicación. La descripción del plan de pruebas llevado a cabo se encuentra en el capítulo 6 de este documento.

Finalmente, en el capítulo 7, se analiza el proyecto desarrollado. Para ello, se utilizan los objetivos propuestos en este capítulo y se determinan los puntos fuertes y los puntos débiles del proyecto. Además, se proponen varias líneas futuras de desarrollo que se consideran interesantes en el caso de que algún desarrollador desee continuar con la línea de investigación propuesta en este proyecto.

Capítulo 2

Estado del arte

En este capítulo se presentarán los contenidos teóricos sobre los que descansan los conceptos básicos de este proyecto. Debido a esto, en este capítulo se desarrollarán los siguientes puntos:

- Estudio de las tecnologías actuales que proporcionan TV a dispositivos móviles.
- Estudio de la arquitectura *IMS*.
- Estudio de las implementaciones SIP que se pueden encontrar en Internet.
- Estudio de los reproductores disponibles para dispositivos móviles.

2.1. Plataformas de TV para dispositivos móviles

En esta sección se describirán las principales tecnologías de TV disponibles para dispositivos móviles. Las tecnologías que se analizarán serán *Digital Video Broadcasting - Handheld*, *DVB-H* [6], *Multimedia Broadcast Multicast Service*, *MBMS* [7] y el estándar de *IPTV* [4].

2.1.1. Digital Video Broadcasting - Handheld

La tecnología DVB-H constituye una plataforma de difusión IP orientada a terminales portátiles, basada en la tecnología *Digital Video Broadcasting - Terrestrial*, o *DVB-T* [8], estándar utilizado por la *Televisión Digital Terrestre* o *TDT*.
Las características más importantes de esta tecnología son las siguientes:

1. **Adaptación de la calidad de la señal**: Esta característica permite la adaptación del vídeo a la resolución específica del dispositivo, ya que estos dispositivos disponen de una pantalla de menor dimensiones. Además, una de las ventajas de que DVB-H sea compatible, o esté basado, en DVB-T es que se puede utilizar la misma banda de frecuencia para emitir información por ambas tecnologías. Esto permite a los operadores trabajar en DVB-H sin necesidad de adaptar su infraestructura.

2. **Reducción del consumo**: Ya que los dispositivos móviles disponen de una batería, esta nueva tecnología debe de reducir el consumo de la misma para evitar su desgaste y deterioro. Para cumplir con este objetivo, la tecnología DVB-H utiliza un procedimiento llamado *time-slicing*. Este procedimiento se basa en la idea que los datos recibidos son enviados a ráfagas en diferentes intervalos de tiempo. Teniendo en cuenta esto, mientras que no estén recibiendo los datos, el sintonizador está inactivo y, por consiguiente, el consumo de batería queda reducido. El procedimiento de *time-slicing* es importante para reducir el consumo de batería, ya dicho consumo se puede reducir hasta un 90 %, pero además es importante para realizar el *Handover*.

3. **Utilización del sistema Handover**: Como ya es bien sabido, el sistema de comunicaciones está basado en diferentes estaciones base que dotan de cobertura a una determinada zona geográfica llamada celda. Mientras un dispositivo está dentro de una celda recibirá los datos de la estación base de dicha celda; sin embargo, a medida que el dispositivo se vaya alejando de la estación base, la calidad de la señal disminuye hasta llegar al punto en que la calidad de la señal será insuficiente para proporcionar el servicio a unos niveles de calidad aceptables. El sistema de *Handover* permite el escaneo de señales de otras estaciones base, para que llegado el momento en el cual la calidad de la señal recibida es muy baja permitir al dispositivo cambiar de celda y seguir recibiendo los datos desde otra estación base, evitando la pérdida de calidad. La utilización que hace la tecnología *DVB-H* del sistema de Handover utiliza el concepto de *time-slicing* explicado en el párrafo anterior. La idea es que en los períodos de tiempo en los cuales el dispositivo no está recibiendo ningún dato, se estén escaneando las señales de otras estaciones base, para que si se presenta el momento de calidad mínima no se tenga que buscar o escanear todas las señales que se están recibiendo.La imagen 2.1 muestra el sistema *Handover*, en dicha imagen vemos la trayectoria del dispositivo móvil a través de tres celdas. En el punto 1, el dispositivo se encuentra conectado a la estación número 1.

Cuando el dispositivo se encuentra en el punto 2, sigue estando conectado a la estación 1, pero percibe la señal de la estación 3, por lo que agrega esta estación como posible estación base si la calidad del servicio empeora. En el punto 3, el dispositivo se encuentra fuera de la cobertura de la estación 1, por lo que en este momento, el dispositivo se encuentra conectado a la estación 3 aunque el usuario no ha percibido el cambio de conexión. Del mismo modo que en el punto 2, en el punto 4 el dispositivo está conectado a la estación base 3, pero agrega a la estación base 2. Finalmente, en el punto 5, el dispositivo se encuentra bajo la cobertura de la estación base 2 y está conectado a dicha base.

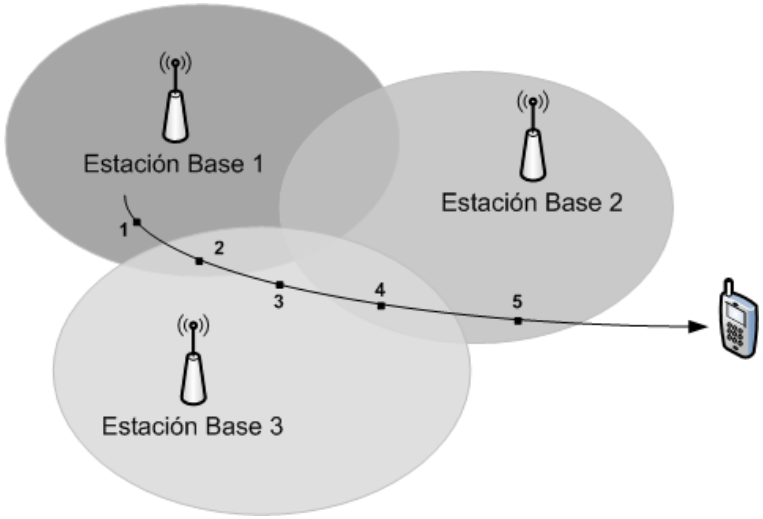

Figura 2.1: Descripción del sistema de Handover.

2.1.2. Multimedia Broadcast Multicast Service

Multimedia Broadcast Multicast Service, *MBMS*, es una tecnología que utiliza las redes diseñadas para *Universal Mobile Telecommunications System*, o *UMTS*, y *Global System for Mobile communications*, o *GSM* [9]. La gran ventaja de este hecho es que no es necesario invertir en la infraestructura que permita entregar al usuario el servicio solicitado.

Este sistema puede ser utilizado para reproducir videos y sonidos pequeños o ligeros, a pesar que el *streaming* también es soportado por esta tecnología. Para la distribución de contenidos más grandes existen otras

tecnologías más recomendables como *DVB-H*, que ha sido explicada en la sección anterior. *MBMS* se utiliza para envíos masivos de servicios de televisión.

Como su propio nombre indica, esta tecnología tiene dos modos de ejecución:

1. **Modo Broadcast**: En este modo, el contenido del servicio es enviado a todos los usuarios posibles que se encuentran dentro del área de servicio de *broadcast*. Las ventajas de este modo es que permite proporcionar los servicios a un mayor número de usuarios. Los receptores de este modo no podrán corregir los posibles errores que se pueden producir en los datos recibidos; aunque sí podrán determinar si existe una pérdida de datos.

2. **Modo Multicast**: La principal diferente entre este modo y el modo anteriormente descrito radica en que los usuarios de un servicio *multicast* deben de estar subscritos a dicho servicio. Por lo tando este modo permite enviar el servicio a todos los usuarios que se encuentran subscritos al servicio.

Si se desea profundizar en esta tecnología se recomienda la lectura [10].

2.1.3. Estándar IPTV

La tecnología *IPTV*, o *Televisión sobre el protocolo IP*, es una tecnología que relevará al actual sistema de televisión. La principal característica de esta tecnología es el uso de la banda ancha sobre el protocolo IP para hacer llegar a los usuarios los contenidos multimedia.
Por otro lado, los contenidos se emitirán cuando el cliente lo solicite, por lo que se proporciona un servicio de *vídeo bajo demanda*, o *View Over Demand*, *VOD*. Esto permite el desarrollo del pago por visión o *Pay Per View*, *PPV*.

En el año 2007 se creó el *Open IPTV Forum* cuyo objetivo es el desarrollo de una especificación *IPTV* que permita la utilización "plug and play"de dispositivos.

2.2. Subsistema Multimedia IP

En esta sección se describirán las principales características y la arquitectura del Subsistema Multimedia IP, también llamado *IMS* de *IP Multimedia Subsystem* [3].

2.2.1. Introducción

El Subsistema Multimedia IP es la parte fundamental de las redes de próxima generación, también llamadas 4G o NGN por *Next Generation Networking* [11]. Esto es debido a que IMS pretende proporcionar los servicios que ofrece Internet utilizando tecnologías móviles.

Sin embargo, no es difícil darse cuenta de que los actuales terminales de tercera generación ofrecen, también, los servicios que proporciona Internet. Las diferencias que hacen que IMS sea el sucesor de los servicios 3G, cuando ambos ofrecen el mismo servicio, son las siguientes:

- **Calidad del Servicio (QoS)**: El actual sistema 3G no ofrece garantías, por parte de la red, sobre el ancho de banda que el usuario puede conseguir, ni tampoco de que el nivel conseguido se mantenga durante toda la sesión. En cambio, IMS cuenta con una provisión de QoS que se asegura de proporcionar mientras dure la sesión.

- **Método de facturación**: IMS proporciona información sobre el servicio que se está utilizando y deja en manos del operador la manera de facturar dicho servicio. Esto ayuda a que los operadores tengan diferentes modelos de facturación dependiendo del servicio que se está utilizando. El modelo de facturación actual de los sistemas 3G tiene en cuenta, únicamente, el número de bytes transmitidos independientemente de su contenido; esto provoca que determinados servicios no sean interesantes para el usuario desde el punto de vista económico. Un ejemplo de mejor facturación de servicios puede ser que se facturen las sesiones multimedia dependiendo de su duración, o que se cobren los mensajes instantáneos por unidad de mensaje.

- **Servicios integrados**: Dada la arquitectura y características de IMS, el coste de desarrollar un nuevo servicio es muy reducido en comparación con el coste que tendría el desarrollo de dicho servicio en un sistema 3G. Además, un sistema IMS permite incluir servicios de terceras personas sin demasiada dificultad. Este hecho es muy importante, ya que

permite a los operadores integrar diferentes servicios pertenecientes a varios desarrolladores.

2.2.2. Requisitos

Una vez descritas las características de un sistema IMS, se describirán los requisitos que debe cumplir dicho sistema. Estos requisitos fueron definidos por 3GPP, *3rd Generation Partnership Project*, en [12]. En dicho documento, IMS es definido como un esquema arquitectónico cuyo objetivo es proporcionar servicios multimedia IP a los usuarios. Además, este esquema debe cumplir las siguientes restricciones:

- **Debe soportar el establecimiento de sesiones multimedias IP**. El objetivo es el servicio de sesiones multimedia sobre redes de conmutación de paquetes, que son las que utiliza IMS. Se entiende por *multimedia*, la existencia de audio y vídeo.

- **Debe disponer de técnicas para la negociación de la Calidad del Servicio (QoS)**. La Calidad del Servicio, *QoS*, se determinad teniendo en cuenta varios factores, como el máximo ancho de banda que un usuario puede obtener dependiendo, por ejemplo, del tipo de subscripción que tenga o del estado de la red. Esto permite que los operadores controlen dicho QoS o que diferencien grupos de usuarios.

- **Debe soportar *roaming*, o migración**. El sistema IMS debe permitir la migración de sus usuarios a diferentes redes de diferentes países sin que ello afecte a las sesiones multimedias que tienen establecidas.

- **Debe ser capaz de trabajar con Internet y con redes de conmutación de circuitos**. La idea de que el sistema IMS deba tener capacidad para trabajar con Internet es importante porque de esta manera se incrementan tanto el número de iniciadores, como de destinatarios de servicios multimedia. Además, como se ha dicho anteriormente, IMS utiliza redes de conmutación de paquetes; este factor junto con las características del *roaming*, o migración, y las políticas de cada país en lo referente a llamadas de emergencia, obligan al sistema IMS a que deba poder funcionar sobre redes de conmutación de circuitos. Un ejemplo de dicha red es la red clásica de teléfono. Cuando el terminal IMS realice una llamada de emergencia utilizando IMS, la red a la que intenta acceder le indicará que deberá hacer dicha llamada utilizando la red de conmutación de circuitos. En ese momento el terminal deberá cambiar a dicha red y efectuar la llamada.

- **Debe de ofrecer control sobre los servicios proporcionados a los usuarios**. Este requisito se centra en las restricciones que puede establecer un operador sobre los servicios invocados por los usuarios. Un sistema IMS debe permitir a los operadores establecer restricciones generales a todos los usuarios, como limitar el ancho de banda dependiendo de los codecs de audio, o restricciones particulares a usuarios o grupos de usuarios que dependan del tipo de subscripción que estos tienen.

- **Debe proporcionar soporte para la creación rápida de servicios sin estandarizar**. Una característica de los sistemas IMS es el rápido desarrollo de servicios, esto se debe a que 3GPP en [13], estandariza las capacidades del servicio en lugar del servicio en sí. Esto provoca que los servicios se desarrollen más rápido y tarden menos tiempo en salir al mercado; y además, que los operadores puedan integrar y proporcionar servicios creados por diferentes desarrolladores.

2.2.3. Arquitectura

En este apartado vamos a hablar de la arquitectura de los sistemas IMS. Dicha arquitectura está organizada por capas, como se puede ver en la figura 2.2. La arquitectura dispone de tres capas: Capa de Aplicación, Capa de Control y Capa de Usuario. El diseño por capas ayuda a minimizar las dependencias entre capas y facilita la adición de nuevas funciones en cualquier capa.

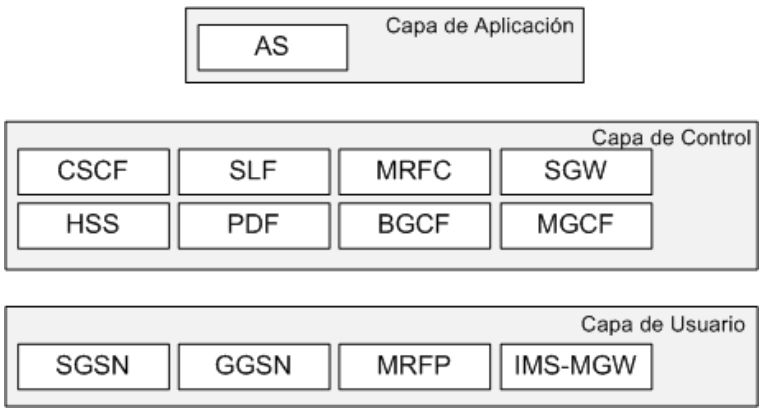

Figura 2.2: Estructura por capas de la arquitectura IMS.

El objetivo de cada capa es el siguiente:

- **Capa de Aplicación**: En esta capa tienen cabida todas las aplicaciones que la red proporcionará a los usuarios. El proceso de integración de una nueva aplicación es bastante rápido ya que IMS define la interfaz que debe de tener dicha aplicación, entendiendo por interfaz el medio de comunicación de la aplicación con el resto de módulos de la red.

- **Capa de Control**: Toda la información de los usuarios, las subscripciones de estos y las sesiones que tienen abiertas se encuentran almacenadas en esta capa. Además, se controlan las sesiones que se han establecido, se administran los recursos multimedia y se controlan las políticas del operador y de seguridad.

- **Capa de Usuario**: En esta tercera capa, se establecen los mecanismos que permiten el acceso a la red a través de diferentes puntos de acceso. Al igual que en la Capa de Aplicación, el diseño por capas permite la inclusión de nuevos módulos de acceso a la red en un futuro sin dificultad.

Como ya se ha indicado, IMS especifica las interfaces que debe de tener cada módulo para que se pueda comunicar con el resto de módulos del sistema. Esta característica junto con el diseño por capas de la red, hacen que la ampliación de la red mediante la creación de nuevos módulos, se realice de una manera rápida, fácil e independiente de los módulos ya disponibles.

Una vez definidas las capas que componen una arquitectura IMS, se describirán las diferentes funciones que componen el sistema. Al hablar sobre la arquitectura IMS se habla de funciones y no de nodos, para dotar a los desarrolladores de cierta independencia a la hora de diseñar la red. De esta manera, se pueden combinar dos funciones en un mismo nodo, o asignar un nodo a cada función, o dejarlo al libre diseño del desarrollador.

Componentes arquitectónicos de IMS

En esta sección se describirán las diferentes funciones que conforman la arquitectura IMS.

Call Session Control Functions (CSCF)

El Call Session Control Functions, CSCF, se compone de tres funciones que son Proxy-CSCF, Interrogating-CSCF y Serving-CSCF. Para simplificar el diseño de la figura 2.2, se han agrupado estas tres últimas funciones bajo el nombre de CSCF; sin embargo, la figura 2.3 desglosa el concepto de CSCF en P-CSCF, I-CSCF y S-CSCF. Ambas imágenes son correctas sólo

que la figura 2.3 muestra una arquitectura más detallada de la función CSCF.

También ha de aclararse que la figura 2.3, muestra la arquitectura de un sistema IMS cuando la red de acceso es *General Packet Radio Service*, *GPRS*. Esta red no es la única compatible, ya que se puede acceder a la red a través de *Asymmetric digital subscriber line*, *ADSL* [14], fibra óptica, Wifi o Wimax, entre otras. De hecho, la aplicación que se desarrolla en este proyecto accederá a la red utilizando una red Wifi. Con todo lo expuesto, se pretende explicar la independencia que tiene una red IMS sobre la red de acceso.

A continuación se procederá a explicar las funciones P-CSCF, I-CSCF y S-CSCF. Todas estas funciones son utilizadas durante el registro de un usuario y el establecimiento de una sesión.

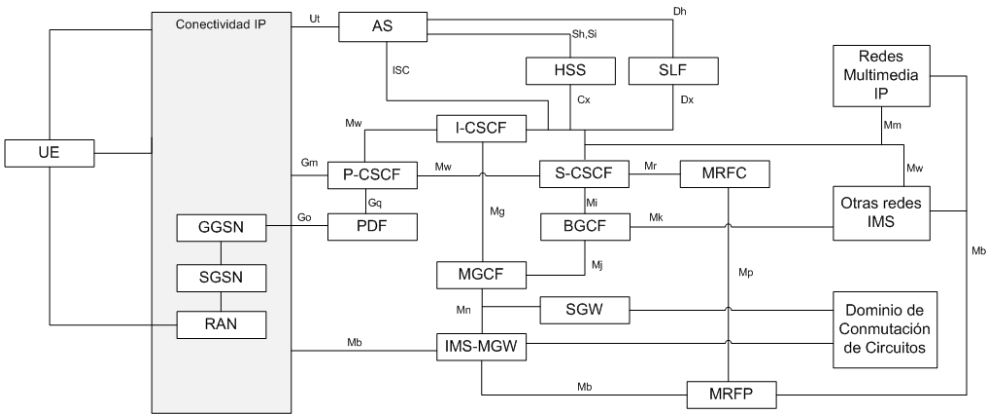

Figura 2.3: Funciones de la arquitectura IMS.

Proxy Call Session Control Function (P-CSCF)
El P-CSCF es el punto de contacto entre el usuario y la red IMS. Todo mensaje que envía el usuario va dirigido a esta función y, a su vez, toda contestación de la red es enviada por el P-CSCF.

Las tareas que debe desempeñar esta función son las siguientes:

- **Compresión SIP**: *Session Initiation Protocol*, *SIP*, es el protocolo utilizado para las comunicaciones con la red IMS y está definido en [15]. Este protocolo está basado en texto y contiene numerosas cabeceras, parámetros e información de seguridad, lo que provoca que los mensajes sean bastante largos. Por esta razón y para acelerar el establecimiento

de sesiones, se pueden comprimir los mensajes que se intercambien entre el usuario, UE, y el P-CSCF. Por lo tanto, el P-CSCF debe ser capaz de comprimir y descomprimir mensajes SIP si el usuario lo especifica.

- **Negociación de las Asociaciones de Seguridad de IP Security o *IPSec* [16]**: Con el objetivo de proteger el tráfico entre el P-CSCF y el UE, este nodo abre un túnel entre él y el usuario. En el establecimiento de dicho túnel se deben gestionar las Asociaciones de Seguridad, SA, de IPSec y se debe aplicar integridad y confidencialidad a los mensajes SIP.

- **Establecer comunicación con Policy Decision Function (PDF)**: El P-CSCF debe mantener la comunicación con el PDF, para manejar información relativa a las políticas que se le aplican al usuario y la forma de facturación de la sesión.

- **Detección de las llamadas de emergencia**: Las llamadas de emergencia utilizan un sistema de conmutación de circuitos mientras que IMS utiliza el sistema de conmutación de paquetes. Esto provoca que si un usuario quiere realizar una llamada de emergencia utilizando la red IMS, el P-CSCF debe de ser capaz de detectar dicha llamada y desviarla a una red de conmutación de circuitos.

Interrogating Call Session Control Function (I-CSCF)

Un usuario de IMS puede subscribirse a diferentes servicios soportados por la red. El punto de contacto de los usuarios subscritos a un determinado servicio y el operador de la red es el I-CSCF, cuyas tareas son las siguientes:

- **Obtener el nombre del siguiente elemento de destino**: Como ya se ha dicho, el primer mensaje se envía al P-CSCF, pero rara vez es éste el destinatario del mensaje. Por ello, el I-CSCF se encarga de descubrir al siguiente destinatario del mensaje. Dicho destinatario puede ser un Serving-CSCF o una aplicación, AS. Esta información la obtiene del Home Subscriber Server, HSS, que se describirá más adelante.

- **Asignar un S-CSCF**: Cuando el usuario se está registrando en la red o dicho usuario no está registrado pero está recibiendo la invitación a un servicio que no requiere autenticación, el I-CSCF debe asignar un S-CSCF para dicho usuario.

- **Encaminar las peticiones hacia un S-CSCF o hacia un servidor de aplicación (AS)**: Cuando la petición la realiza la red, el AS, el I-CSCF enviará la petición hacia el S-CSCF. En caso de que el usuario sea el que realice la petición, el I-CSCF mandará la petición al AS.

- **Proporcionar funcionalidad Topology Hiding Inter-Network Gateway (THIG)**:La funcionalidad THIG permite esconder la configuración, capacidad y topología de nuestra red a otras redes externas. Para ocultar esta información THIG debe encriptar y desencriptar todas las cabeceras que contengan información sobre la topología.

Serving Call Session Control Function (S-CSCF)
El S-CSCF es el responsable de la administración de registros, toma decisiones sobre el encaminamiento de los mensajes, administra el estado de las sesiones y almacena el perfil de los servicios.

El proceso de autenticación de un usuario en una red IMS utiliza el estándar *HTTP-Digest* [17]. Siguiendo este estándar cuando un usuario quiere registrarse, la petición le llega al S-CSCF el cual descarga del HSS los datos de autenticación. Basado en dichos datos, envía un reto al usuario. Después de recibir la respuesta y verificarla, el S-CSCF acepta el registro y supervisa el estado de dicho registro. Una vez que este proceso termine, el usuario podrá iniciar y recibir sesiones IMS.

El perfil de un servicio es una colección de información del usuario que se almacena en el HSS. El S-CSCF descarga el perfil del servicio asociado a un usuario en concreto cuando dicho usuario se registra en la red. La información incluida en dicho perfil permite conocer a qué AS debe enviar la petición y también permite conocer información acerca de las políticas asociadas a dicho usuario.

Además, el S-CSCF toma decisiones de encaminamiento. Cuando recibe una petición del usuario a través del P-CSCF, el S-CSCF debe de comprobar si tiene posibilidad de contactar con el AS o enviar la petición a una red de conmutación de circuitos o a otras redes IP.

Home Subscriber Server (HSS)
El Home Subscriber Server es una base de datos que contiene información de los usuarios. Esta base de datos contiene toda la información relacionada con las subscripciones de los usuarios y que será utilizada para manejar las sesiones multimedia. Dicha información contiene la identidad de los usuarios, información del registro que dicho usuario tiene, parámetros de acceso a la red e información de los servicios que puede lanzar el usuario. La identidad del usuario se compone de una identidad privada, asignada por la red de origen y que se utiliza para registrarse y autenticarse, y la identidad pública que se utiliza para que otros usuarios de la red puedan establecer

sesiones con este. Los parámetros de acceso permiten establecer sesiones y contiene información como el S-CSCF asociado a dicho usuario o si tiene permisos para hacer *roaming*. Finalmente, la información de los servicios indica qué servicios puede solicitar el usuario.

Subscription Locator Function (SLF)
Si el número de subscripciones de los usuarios es pequeño, es posible que solo se necesite un HSS; pero en caso contrario se necesitarán dos o más HSS. En esta última situación, la red necesitará los Subscription Locator Function, SLF. Son simples bases de datos que mapean a todos los usuarios dentro de los diferentes HSS. Una función que haga una petición a un SLF con la dirección de un usuario, será contestado con la dirección del HSS que contiene la información de dicho usuario.

Application Server (AS)
En la capa de Aplicación se encuentra el Application Server, AS, cuya finalidad es almacenar y ejecutar los servicios que ofrece la red IMS. Las funciones de un AS son las siguientes:

- Procesar sesiones SIP
- Crear respuestas SIP
- Enviar información sobre la sesión a las funciones de tarifación

Un AS debe de ser capaz de procesar y crear mensajes SIP ya que el AS se conecta al S-CSCF y a través de dicha conexión utiliza SIP como protocolo de comunicación.
Por otro lado, como podemos observar en la figura 2.4, existen tres tipos diferentes de AS, los cuales son:

- **SIP Application Server (SIP AS)**: Este es el servidor de aplicaciones que almacena y ejecuta servicios multimedia IP. Estos servidores puede proporcionar servicios de presencia, de mensajería, de conferencia o de *Push to Talk* sobre dispositivos móviles.
- **Open Service Access-Service Capability Server (OSA SCS)**: El uso de servicios OSA proporciona control de llamada, control de los datos de sesión, gestión de las políticas y costes de desarrollo de servicios. Además, puede ser utilizado como un mecanismo estandarizado para utilizar AS desarrollados por terceras personas dentro de la red IMS de una manera segura. OSA SCS usa la *OSA Application Program Interface* para comunicarse con OSA AS.

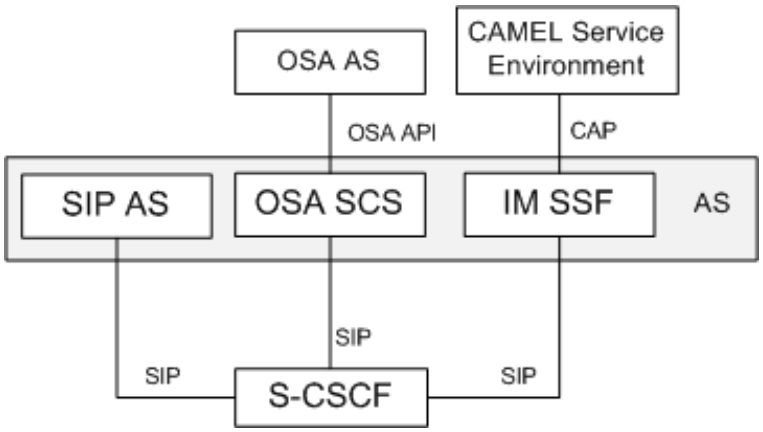

Figura 2.4: Relación entre los diferentes tipos de AS.

- **IP Multimedia Service Switching Function (IM-SSF)**: Este servidor proporciona servicios desarrollados por el Entorno de Servicio CAMEL, CSE, que en un principio fueron desarrollados para GSM. IM-SSF almacena características de las redes CAMEL y se comunica con dichas redes utilizando la interfaz *CAMEL Application Part* o *CAP* [18].

Media Resource Function Controller (MRFC) y Media Resource Function Processor (MRFP)
El Media Resource Function Controller, MRFC, y el Media Resource Function Processor, MRFP, proporcionan mecanismos para el transporte de servicios como el servicio de conferencia o el servicio de anuncios a los usuarios. El MRFC actúa como un agente de usuario SIP y contiene una interfaz SIP con S-CSCF. Además, controla los recursos del MRFP utilizando una interfaz H.248. Más información sobre esta interfaz en .....
Por otro lado, el MRFP implementa todas las funciones relacionadas con los medios, entendiendo por *medio* audio o video. Estas funciones pueden ser mezclar medios o reproducirlos.

Breakout Gateway Control Function (BGCF)
Como ya se ha explicado anteriormente, cuando el usuario realiza una llamada de emergencia utilizando la red IMS es la propia red la que debe de identificar la llamada de emergencia y desviarla a una red de conmutación de circuitos. Cuando el S-CSCF decide desviarla, envía la sesión SIP al *Breakout Gateway Control Function*, BGCF, y este último decide si enviarla a la misma red en la que se encuentra o a una red externa. Si decide enviar

la sesión a la misma red, entonces el BGCF elige un *Media Gateway Control Function*, MGCF, que maneje la sesión más adelante. En cambio, si decide enviar la sesión a otra red externa, el BGCF manda la sesión a otro BGCF de la red de destino.

Media Gateway Control Function (MGCF)
La función del *Media Gateway Control Function*, MGCF, consiste en realizar una conversión del protocolo utilizado en la capa de aplicación, SIP. Esto se debe a que cuando el MGCF realiza su función significa que hay una llamada de emergencia que se debe tratar, y dicha llamada terminará en un dominio de conmutación de circuitos. Pero dadas las características de la red IMS, la llamada no se puede derivar a la nueva red sin tratarla antes. Este tratamiento consiste en convertir los mensajes SIP a mensajes *ISDN User Part*, ISUP. Esta conversión puede observarse en al figura 2.5 . Una vez realizada la conversión, envía el resultado al *Signalling Gateway*, SGW. Además el MGCF controla el *IMS Media Gateway*, IMS-MGW.

Signaling Gateway (SGW)
El *Signaling Gateway*, SGW, realiza la conversión a nivel de transporte. Su función consiste en convertir los paquetes M3UA en MTP3, los paquetes SCTP en MTP2, y las cabeceras IP en L1, como se puede observar en la figura 2.5.

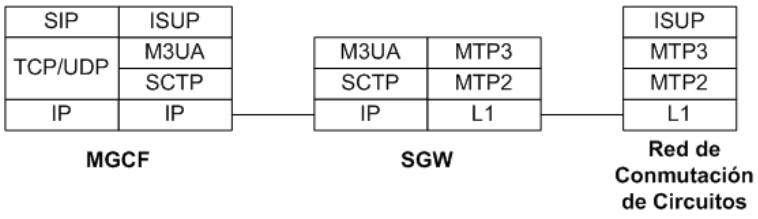

Figura 2.5: Conversión en SGW.

IMS Media Gateway (IMS-MGW)
El *IMS Media Gateway*, IMS-MGW, proporciona un enlace entre las redes de conmutación de circuitos e IMS. Además realiza la conversión de los tipos de medios entre la red IMS y la red de conmutación de circuitos, y proporciona mensajes para los usuarios de la red de conmutación de circuitos.

Policy Decision Function (PDF)
El *Policy Decision Function*, PDF, es el encargado de gestionar el QoS que

un usuario dispondrá durante la sesión. Será utilizado cuando se necesite tomar una decision de política local. En este caso, el P-CSCF le enviará al PDF información del usuario y el *Session Description Protocol*, SDP, que contiene información de los medios que solicita el usuario. El PDF analiza estos datos y devuelve una decisión de autorización, que llegará al equipo del usuario, UE.
Además de esto, el PDF recibe información como las veces que el usuario recupera la conexión y recibe los medios solicitados, o si el usuario está recibiendo los datos en *streaming* o por el modelo tradicional. Esta información se la envía el PDF al P-CSCF, para que éste tome las medidas de facturación oportunas.

Serving GPRS Support Node (SGSN)
El *Serving GPRS Support Node*, SGSN, enlaza el *Radio Access Network*, RAN, con la red de conmutación de paquetes. Sus dos funciones principales son las siguientes:

- **Gestión de la movilidad**: La gestión de la movilidad está relacionada con la localización y el estado del UE. Además autentica tanto al UE como al subscriptor.

- **Gestión de la sesión**: La gestión de la sesión se ocupa del mantenimiento de la conexión.

El SGSN funciona como pasarela entre el UE y el GGSN, y por tanto otra tarea consiste en asegurarse de la recepción del QoS definida al inicio de la sesión.

Gateway GPRS Support Node (GGSN)
El *Gateway GPRS Support Node*, GGSN, proporciona la operatividad con redes externas basadas en paquetes. Su principal función consiste en conectar al UE con estas redes donde residen los servicios y las aplicaciones basadas en IP. El GGSN encamina los mensajes SIP del UE hacia el P-CSCF y viceversa. Además, supervisa el uso del *Policy Decision Point*, PDP, y proporciona información para la tarifación.

Una vez explicadas todas las entidades que forman la arquitectura IMS, se van a explicar los diferentes puntos de referencia que conectan todas las entidades ya descritas. Los diferentes puntos de referencia se pueden observar en la figura 2.3.

Puntos de referencia

Una vez que se han descrito todas las funciones que forman parte de la arquitectura IMS, se describirán los diferentes puntos de referencia que permiten la comunicación entre las funciones ya descritas.

Punto de referencia Gm

El punto de referencia Gm conecta el UE a la red IMS y transporta todos los mensajes SIP. Las funciones de este punto de referencia son los siguientes:

- **Registro**: Cuando el usuario se quiere registrar, este punto de referencia envía la solicitud de registro con los mecanismos de seguridad soportados al P-CSCF. Además durante el proceso de registro, el UE intercambia los parámetros necesarios para autenticarse en la red y autenticar la propia red, y negocia la los parámetros de seguridad IPSec.
- **Control de sesión**: Estos procedimientos contienen mecanismos para el control de sesiones iniciadas y no iniciadas por el UE. Este punto de referencia une el UE con el P-CSCF.
- **Transacciones**: Através de este punto de referencia, se pueden establecer conversaciones que no necesiten diálogo como el envío del mensaje SIP MESSAGE y la recepción de la correspondiente respuesta.

Punto de referencia Mw

Los procedimientos de este punto de referencia pueden ser agruados en 3 categorías:

- **Registro**: En el proceso de registro el P-CSCF usa este punto de referencia para enviar la petición de registro del UE al I-CSCF. Y este último, utiliza el mismo punto de referencia para enviar dicha petición al S-CSCF.
- **Control de sesión**: Estos procedimientos contienen mecanismos para el control de sesiones iniciadas y no iniciadas por el UE. Con este punto de referencia se une el P-CSCF al S-CSCF y el S-CSCF con el I-CSCF.
- **Transacciones**: Los procedimientos de transacciones son iguales a los procedimientos de transacciones que posee el punto de referencia Gm, explicado anteriormente.

Punto de referencia ISC

Como ya se ha explicado anteriormente, en la arquitectura IMS los AS son

entidades que almacenan y ejecutan servicios. Por lo tanto, debe de existir algún punto de referencia que permita enviar y recibir mensajes SIP entre el CSCF y el AS. Este punto de referencia se llama *IMS Service Control*, ISC, y el protocolo de comunicación que utiliza es SIP. Las funciones de dicho punto de referencia son las siguientes:

- **Conexión con el AS**: Cuando un S-CSCF recibe una petición SIP, debe analizarla y una vez hecho esto debe decidir si enviar dicha petición al AS para que la procese.

- **Conexión con el S-CSCF**: Una vez que el AS recibe la petición SIP, dicha entidad tiene que ser capaz de responder a dicho mensaje con otro mensaje que llegue hasta el S-CSCF.

Punto de referencia Cx

El HSS contiene información de los usuarios y en ocasiones el CSCF necesita acceder a dicha información. El punto de referencia Cx permite al CSCF acceder a la información almacenada en HSS. Dicho punto de referencia utiliza el protocolo *Diameter*. Las funciones de dicho punto de referencia son:

- Acceso a la información de los registros de dicho usuario.

- Acceso a la información de localización.

- Mantenimiento de los datos relativos al perfil del usuario.

- Transmisión segura al S-CSCF de los datos relativos a la autenticación del usuario.

Punto de referencia Dx

Cuando existen dos o más HSS, en la red IMS exiten uno o varios SLF que deben mapear a los diferentes usuarios dentro de todos los HSS que posee la red. Este punto de referencia es el que une el/los SLFs con el I-CSCF y el S-CSCF. Este punto de referencia utiliza el protocolo de comunicación *Diameter*.
Para conseguir la dirección de un HSS, el I-CSCF o el S-CSCF envían al SLF un *Location-Information-Request*, LIR, y recibirán un *Location-Information-Answer*, LIA. La figura 2.6 muestra este proceso.

Punto de referencia Sh

Puede ser que un AS necesite conocer la información de algún usuario o necesite conocer a qué S-CSCF debe enviar las peticiones SIP. Este tipo de información está almacenada en el HSS, por lo que debe de existir un punto

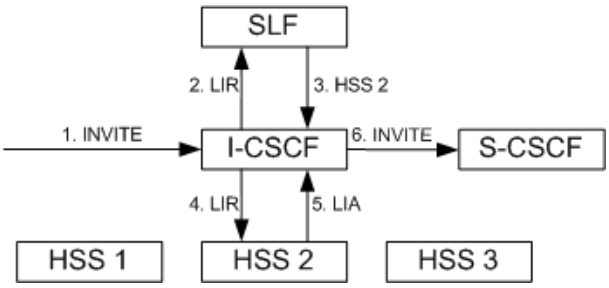

Figura 2.6: Resolución del HSS utilizando SLF.

de referencia que permita a los AS acceder a esta información en el HSS. El punto de referencia es el Sh y utiliza el protocolo *Diameter*. Este punto de referencia permite acceder a los datos de los usuarios y a sus subscripciones y notificaciones.

Punto de referencia Si
Cuando el AS es un CAMEL AS (IM-SSF), utiliza este punto de referencia para comunicarse con el HSS. Este punto de referencia se usa para enviar información de las subscripciones CAMEL utilizando el protocolo *Mobile Application Part*, MAP.

Punto de referencia Dh
Cuando la red posee diferentes HSS, los AS no saben a qué HSS necesitan interactuar. Por tanto, los AS deben contactar primero con el SLF utilizando este punto de referencia, Dh. Este punto de referencia utiliza el protocolo Diameter y su objetivo es proporcionar los mecanismos de encaminamiento proporcionados por un agente de redirección Diameter.

Punto de referencia Mm
Este punto de referencia une la red IMS con otras redes IMS. A través de dicho punto de referencia el I-CSCF recibirá peticiones de sesión de otro servidor SIP o terminal y el S-CSCF enviará las peticiones de sesiones a otras redes IMS.

Punto de referencia Mg
Este punto de referencia permite unir las redes de conmutación de circuitos con el I-CSCF, a través del MGCF. El protocolo utilizado por este punto de referencia es SIP. Cabe recordar, que la función del MGCF consiste en convertir las cabeceras SIP en ISUP y viceversa.

Punto de referencia Mi
Cuando un S-CSCF descubre que la sesión necesita utilizar una red de conmutación de circuitos, utiliza este punto de referencia para mandar la sesión al BGCF.

Punto de referencia Mj
El punto de referencia Mj, es utilizado por el BGCF para enviar una sesión a un dominio de conmutación de circuitos dentro de la propia red. Este punto de referencia utiliza el protocolo de comunicación SIP.

Punto de referencia Mk
Si la red de conmutación de circuitos a la cual derivar la sesión está localizada en una red externa, el BGCF utiliza el punto de referencia Mk. Dicho punto de referencia utiliza el protocolo de comunicación SIP.

Punto de referencia Mn// El punto de referencia Mn controla el plano de usuario entre el acceso IP y el IMS-MGW. También controla el plano de usuario entre los dominios de conmutación de circuitos y el IMS-MGW. Este punto de referencia está basado en H.248.

Punto de referencia Ut
Este punto de referencia une el UE con los AS, y permite a los usuarios administrar de forma segura su información relacionada con los servicios de red. Este punto de referencia utiliza *Hypertext Transfer Protocol*, HTTP.

Punto de referencia Mr
Cuando el S-CSCF necesita activar un servicio de transporte, envía un mensaje SIP a MRFP a través del punto de referencia Mr.

Punto de referencia Mp
El punto de referencia Mp es utilizado por MRFC para controlar el stream de algún medio. Este punto de referencia utiliza el protocolo H.248.

Punto de referencia Go
El propósito de este punto de referencia es el de asegurarse que tanto el QoS, como las direcciones de origen y de destino tienen los valores negociados. El protocolo utilizado en este punto de referencia es *Common Open Policy Service*, COPS.

Punto de referencia Gq
El objetivo de este punto de referencia es que el PDF envíe información

sobre las políticas al P-CSCF. Esta información se enviará por cada mensaje SIP que contenga un SDP. Esto asegura que que el PDF envía la información adecuada para autorizar los medios de todos los escenarios de inicio de sesión posibles.

2.2.4. Elementos identificativos dentro de IMS

Una vez que se tiene una visión general de la arquitectura IMS se va a proceder a la explicación de los elementos que se utilizan para identificar a un usuario dentro de una arquitectura IMS. Estos elementos son dos: Identidad Pública de Usuario e Identidad Privada de Usuario, ambas se describirán a continuación.

Identidad Pública de Usuario

Una Identidad Pública del Usuario identifica unívocamente a un usuario de IMS. Esta identidad es proporcionada por el operador y, como mínimo, cada usuario debe poseer una de ellas. Una Identidad Pública de Usuario puede ser tanto una URI SIP [15] como un URI TEL [19]; y ambas son usadas para encaminar mensajes SIP.
El formato de una URI SIP es el siguiente *sip:nombre de usuario@dominio*, aunque los operadores pueden modificar este formato de acuerdo a sus propias necesidades. Por otro lado, el formato de TEL URI es *tel:numero de teléfono.*Puede parecer innecesaria la utilización de TEL URI como elemento identificador, sin embargo, este elemento goza de una relevante importancia para realizar llamadas desde un terminal IMS al sistema telefónico tradicional, PSTN.
Por lo general se suele asociar un SIP URI y un TEL URI por usuario, pero los proveedores pueden ofrecer más para que el usuario pueda diferenciar diferentes cuentas, como por ejemplo, la cuenta personal y la cuenta del trabajo.

Identidad Privada de Usuario

Además de las Identidades Públicas del Usuario, cada usuario dispone de una única Identidad Privada de Usuario. El formato de esta identidad se corresponde con el formato de un Identificador de Acceso a la Red, o *NAI* del inglés *Network Access Identifier* [20], el cual es *sip:nombre de usuario@dominio.*

El objetivo de esta identidad es identificar las subscripciones y realizar tareas de autenticación.

2.2.5. Proceso de Autenticación en IMS

Por último, en esta sección se explicará el proceso de autenticación de un usuario en una red IMS. El proceso de autenticación de un usuario en la red se realiza mediante el uso del Resumen HTTP o *HTTP Digest* [17].

Por lo general, cualquier método de autenticación se basa en un secreto compartido entre ambas partes. Para que una parte se autentique frente a la otra, la primera debe de dar muestras a la segunda de que conoce el secreto compartido. Por lo que todos los problemas de los método de autenticación radican entorno a la idea de cómo transmitir esa prueba de una manera segura.
En este caso, el término "seguro" hace referencia a los conceptos de integridad y confidencialidad:

- **Integridad**: Este concepto se refiere a que cuando un destinatario recibe un mensaje, debe de ser capaz de determinar si dicho mensaje ha sido modificado por terceras personas.
- **Confidencialidad**: El concepto de confidencialidad hace referencia a la imposibilidad de acceder al contenido del mensaje por una tercera persona que tenga acceso a dicho mensaje.

Utilizando el Resumen HTTP, los usuarios son capaces de demostrar, de manera segura, que conocen el secreto compartido sin necesidad de enviar el propio secreto. Para ello, utilizan el secreto compartido, un reto enviado por el servidor, la información del propio usuario y un par de valores que transmiten frescura al proceso en un algoritmo *hash*. Dicho algoritmo no es más que una función unidireccional que transforma cualquier entrada de una longitud variable y la transforma en un resultado de una longitud fija. Por función unidireccional se entiende aquella función que es difícil obtener la entrada a partir de la salida. Ejemplos de este tipo de algoritmos son MD5 [21] y SHA1 [22].

Para terminar con esta sección, decir que si se quiere profundizar en los conceptos y arquitectura de IMS recomiendo la lectura de [23] y [24].

2.3. Implementaciones SIP

En esta sección hablaremos de las diferentes aplicaciones *SIP* que se pueden encontrar en internet. Pero antes de analizar las diferentes aplicaciones se va a analizar el concepto de pila.

Una pila es un conjunto de protocolos que permiten el intercambio de información entre diferentes dispositivos. En una pila de protocolos, cada protocolo ofrece un servicio a los protocolos que se encuentran por encima suya. En el modelo TCP/IP se pueden encontrar las siguientes capas, ordenadas de mayor nivel a menor nivel:

- **Capa de Aplicación**: En esta capa se definen los protocolos que utilizan las aplicaciones para intercambiar datos.
- **Capa de Transporte**: Las principales tareas de esta capa consisten en fraccionar los datos recibidos de la capa de aplicación en partes más pequeñas y hacerlos independientes del tipo de red que se tenga en las capas inferiores. Además, en este nivel se controla la calidad de servicio o el control de flujo.
- **Capa de Internet**: El propósito de esta capa es enviar los paquetes recibidos de las capas superiores hacia el destino, sin importar la ruta que sigan.

Una vez que se ha introducido el concepto de pila, la figura 2.7 representa una pila de protocolos donde se encuentra el protocolo SIP. Como podemos observar, SIP es un protocolo de la capa de Aplicación que puede utilizar tanto *Transmission Control Protocol*, *TCP* [25], como *User Datagram Protocol*, *UDP* [26] y que utiliza IP como protocolo de internet.

2.3.1. GNU osip y eXosip

Este proyecto posee licencia GPL y está escrito en C. Las ventajas de este proyecto son que el programa es muy ligero, en cuanto a peso de la aplicación como a líneas de código, y puede utilizarse tanto en emuladores de dispositivos IP como en programas SIP embebidos.

Las plataformas en las cuales se puede compilar esta librería son las siguientes:

- GNU/Linux.

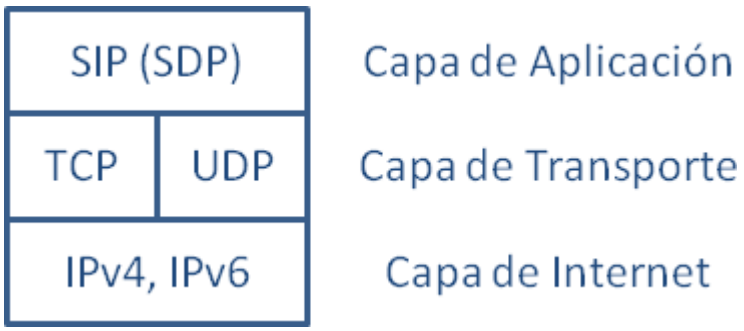

Figura 2.7: Representación de una pila con el protocolo SIP.

- MacOSX (Darwin).
- OpenBsd 3.1 y 3.2. *FreeBSD* y *NetBSD* son necesarios para la compilación.
- Windows NT, 95 y 2000. Se necesita *Visual C++* versión 6.0 o *cigwin*.
- Solaris.
- HP-Unix.
- Sistemas embebidos con linux.

El proyecto GNU osip, [27], comenzó en septiembre del año 2000 junto con el protocolo SIP. Este proyecto está compuesto por un parseador de mensajes SIP y *Session Description Protocol*, *SDP* [28], y una librería para el manejo de transacciones SIP. Para realizar el control de transacciones se basa en una máquina de estados que actuará dependiendo del estado y de las entradas proporcionadas del entorno. Hay que decir que este programa no proporciona una pila completa del protocol SIP, ya que el objetivo es proporcionar el conjunto de características comunes a todos los clientes SIP.

El API utilizado por osip es algo complejo, esto provocó la creación del proyecto osip2, que contiene un API más sencillo. A pesar de este cambio, los proyectos osip son bastante complejos ya que contienen un API bastante extenso y además se pueden construir mensajes mal formados sin que el programa te avise de ello. Este último detalle nos lleva a suponer que el *parser* de los mensajes es morfológico y no estudia la correcta utilización de las cabeceras involucradas en el mensaje.

Actualmente, existe una extensión de la librería osip, llamada eXosip. Esta nueva librería se puede encontrar en [29].

2.3.2. Sofia-SIP

El proyecto Sofia-SIP [30] ha sido desarrollado por centro de investigación de Nokia, *Nokia Research Center*. Este proyecto está escrito en C y la plataforma para la que se ha desarrollado es para GNU/Linux. Entre sus características destacan las siguientes:

- Tiene un soporte para SSL/TLS, para proporcionar cifrado de mensajes y autenticación del servidor y de los clientes.
- Puede ser ejecutado sobre UDP o TCP, y sobre IPv4 [31] o IPv6 [32].
- Soporte completo de una conversación SIP. Esto significa que puede crear y tratar todos los mensajes y cabeceras definidas en [15].
- Tiene soporte para SDP, y para *Session Traversal Utilities for NAT*, *STUN* [33]. Este último protocolo sirve para que los clientes que estén detrás de una *Network address translation*, *NAT*, sepan cuál es su IP pública y características de la red.

Existen numerosas aplicaciones SIP, desarrolladas por el grupo de Nokia, que utiliza el proyecto Sofia-SIP.

2.3.3. Minisip

Este proyecto,[34], permite establecer llamadas telefónicas, enviar mensajes instantáneos y realizar videollamadas a usuarios conectados dentro del misma red SIP. Entre sus características, cabe destacar que es un cliente completo de SIP que soporta el protocolo STUN y *Transport Layer Security*, *TLS* para proporcionar seguridad. Además es un proyecto que al estar desarrollado en C++ puede ser ejecutado en las siguientes plataformas:

- Linux.
- Dispositivos con Linux embebido.
- Windows XP y 2000. Respecto a estas plataformas cabe destacar que para Windows XP el programa no está completamente testeado, mientras que el Windows 2000 la fase de test no ha comenzado todavía.

Actualmente el equipo de desarrolladores están portando y testeando la aplicación para Pocket PC.

2.3.4. Conclusiones sobre las pilas SIP

Nuestra aplicación necesitará una pila SIP para establecer la comunicación con la red IMS. El objetivo de esta sección era el estudio de alguna pila SIP desarrollada en C# y que fuera de código abierto para poder incorporarla a nuestro proyecto. Sin embargo, todos los proyectos presentados en esta sección no pueden ser utilizados dentro de nuestro cliente ya que no están programados en C#.
Por otro lado, se ha encontrado una pila SIP en C# y de código abierto, [35]. Finalmente se desechó la idea de incorporar este proyecto a nuestro cliente por las siguientes razones:

- La pila SIP encontrada posee manejadores para soportar diversos protocolos como STUN, *Hypertext Transfer Protocol*, *HTTP* [36] o *File Transfer Protocol*, *FTP* [37].

- Es una pila completa que contiene todos los tipos de mensajes especificados en [15]; mientras que para el proyecto que nos ocupa sólo necesitaremos un pequeños subconjunto de todos los mensajes SIP.

- El pequeño subconjunto de mensajes SIP que necesitamos provoca que se tarde menos tiempo en el desarrollo de nuestra propia pila SIP que en la integración de la pila encontrada en nuestro código.

Para conocer más detalles sobre el desarrollo del módulo de mensajería de nuestra aplicación, se recomienda la lectura de la sección 3.2.1.

2.4. Reproductores disponibles para Windows CE

El objetivo de esta sección es realizar un estudio sobre los reproductores multimedia que se encuentran disponibles para la plataforma Windows CE. Una vez realizado este estudio, podremos seleccionar uno de estos reproductores para utilizarlo en el proyecto.

2.4.1. Windows Media Player 10 Mobile

Windows Media Player 10, [38], es un reproductor que viene instalado con el sistema operativo. Este reproductor soporta una gran variedad de formatos de vídeo y audio. Además, reproduce archivos multimedia que estén almacenados en el dispositivo o en Internet.

Este reproductor es compatible con los formatos de archivo y Windows Media, WMV.

Los códecs de audio que puede reproducir este programa son los códecs de Microsoft Windows Media, versiones 1.0, 2.0, 7, 8 y 9.Y los códecs de vídeo soportados son los siguientes:

- Códec de vídeo de Microsoft Windows Media, versiones 7, 8 y 9.
- Códec para imagen de vídeo de Microsoft Windows Media 9, versiones 1.0 y 2.0.
- Códec Microsoft MPEG-4, versión 2.0 y 3.0.
- Códec de vídeo ISO MPEG-4, versión 1.0.

2.4.2. HTC Streaming Media

Este reproductor viene instalado con Windows Mobile 6. El objetivo de este programa consiste en reproducir videos de Internet, utilizando una conexión *Real-Time Streaming Protocol*, *RTSP* [39]. El reproductor Streaming Media permite reproducir ficheros 3GP [40] y MPEG-4. Además, permite la reproducción de ficheros SDP.

El principal problema de este reproductor es la falta de documentación que permitiría realizar un estudio más detallado de dicho reproductor.

2.4.3. The Core Pocket Media Player

The Core Pocket Media Player o *TCPMP* [41], es un reproductor en código abierto desarrollado en C++ que permite la reproducción//

Los códecs de audio soportados por este programa son Advanced Audio Coding (AAC), Free Lossless Audio Codec (FLAC), Monkey's Audio (APE), MPEG audio layer-2 (MP2), MPEG audio layer-3 (MP3), Musepack, Ogg Vorbis, True Audio (TTA), WAVEform audio format (WAV) y WavPack. Además soporta reproduce videos codificados con DIVX, H.263, H.264, QuickTime, Windows Media Video y Xvid.

Además de todo lo anterior este reproductor soporta MPEG y MPG.

2.4.4. VideoLan Client

VideoLan Client (VLC), [42], es uno de los reproductores multimedia más conocidos en la actualidad. Este reproductor de código abierto, está desarrollado en C y se encuentra portado a una gran variedad de plataformas como son Linux, Microsoft Windows, Mac OS X, BeOS, BSD y Solaris. Además de esto, es también conocido por la cantidad de formatos [1] que soporta es programa:

- **Contenedores**: 3GP, ASF, AVI, FLV, MKV, QuickTime, MP4, Ogg Vorbis, OGM, WAV, MPEG-2 (ES, PS,TS, PVA,MP3), AIFF, Raw audio, Raw DV y FLAC
- **Formatos de vídeo**: Cinepak, DV, H.263, H.264/MPEG-4 AVC, HuffYUV, Indeo , MJPEG, MPEG-1, MPEG-2, MPEG-4 Part 2, Sorenson, H.263, Flash Video, Ogg Theora, VC-1, VP5, VP6 y WMV
- **Formatos de audio**: AAC, AC3, ALAC, AMR, DTS, DV Audio, FLAC, MACE, MP3, QDM2/QDMC, RealAudio, Speex, Screamtracker 3/S3M, TTA, Vorbis y WMA

El principal problema de este reproductor es que no se encuentra portado a la plataforma Windows CE, por lo que el proceso de migración debe realizarse de manera manual a partir del código fuente disponible en la página oficial de VideoLan.

[1] El soporte de los siguientes formatos y contenedores están sujetos a la plataforma en la cual se utiliza el programa. Esto quiere decir que dependiendo de la plataforma objetivo, el programa puede no soportar algunos de los formatos mencionados.

Capítulo 3

Desarrollo del cliente IPTV

En este capítulo se describirá el desarrollo del cliente de IPTV. En la siguiente sección se hará una descripción del general de dicho desarrollo y se presentarán los diferentes módulos que serán descritos en las siguientes secciones. El objetivo de este capítulo es describir y presentar los aspectos más relevantes del programa.

3.1. Descripción General

El objetivo del proyecto es el desarrollo de un cliente de IPTV para dispositivos móviles. El primer problema con el que nos encontramos al intentar abordar la fase de desarrollo es que no existe un reproductor multimedia que se adecúe a las necesidades de este proyecto[1]. Por esta razón se decide la creación de dos programas:

- **IPTV Client versión para dispositivos móviles**: Este programa es el objetivo del proyecto fin de carrera. Ya que el ámbito en el que este proyecto tiene sentido es el de los dispositivos móviles.

- **IPTV Client versión para ordenador**: Esta versión será una versión estable que sirva para testear la versión para dispositivos móviles. Los resultados de esta versión sólo servirán para hacerse una idea de cómo debe de funcionar el código de la versión para dispositivos móviles. Podemos hacer esta versión ya que para ordenador, sí disponemos de reproductores que satisfacen las necesidades del proyecto.

[1]Para profundizar en los reproductores existentes para dispositivos móviles se recomienda la lectura de la sección 2.4

Una vez que hemos tomado la decisión de crear dos proyectos para diferentes plataformas, nos encontramos con que tenemos dos proyectos prácticamente idénticos, que sólo se van a diferenciar en la plataforma en la que se van a ejecutar, pues incluso las interfaces deben de ser iguales y ambas deben pedir los mismos datos. Esto provoca que el código de la funcionalidad de la aplicación esté replicado, con los problemas que esto conlleva:

- Esta redundancia de código obliga a que toda modificación de dicho código se realice en ambos proyectos. Si cualquier cambio no se realiza en ambos proyectos, tendremos dos proyectos diferentes y, en ese caso, no podremos extrapolar los resultados de la versión para ordenador.
- La fase de documentación de los códigos habría que hacerla dos veces cuando en verdad estamos comentando el mismo código.

Dadas las razones anteriores, llegamos a la conclusión de extraer la funcionalidad de la aplicación a un tercer proyecto. Con esto, eliminaremos redundancias y ahorraremos tiempo.
En las siguientes secciones se describirá el desarrollo de los tres proyectos descritos anteriormente.

3.2. Descripción de la librería *Functionality*

Como ya se ha descrito anteriormente, uno de los proyectos contiene la funcionalidad del cliente IPTV. Esto nos proporciona numerosas ventajas, entre las cuales destacan las siguientes:

1. Con este proyecto tenemos el funcionalidad aislada e independiente de la interfaz y del dispositivo objetivo de la aplicación. Esto nos proporciona mucha flexibilidad ya que podemos portar el código a diferentes dispositivos con gran facilidad, sólo tenemos que definir el nuevo proyecto y diseñar la nueva interfaz.
2. En el caso particular de este proyecto, podemos tener una versión para ordenador y otra para dispositivos móviles con un único código en común. Lo que nos facilita la tarea de modificación de dicha funcionalidad básica, al tener que programar sólo un código y no dos, caso que nos pasaría si tuviéramos únicamente dos proyectos para cada versión.

El siguiente punto de decisión, es cómo crear el proyecto de funcionalidad básica para que pueda ser utilizado por otros proyectos. La manera de hacer esto es definiendo el proyecto como una librería de clases para obtener una

.dll que será utilizado, posteriormente, por los proyectos específicos de cada plataforma.

Una vez definido el tipo del proyecto y el objetivo que debe de tener dicho proyecto, debemos hacer un análisis de la estructura de dicho proyecto, ver los componentes o módulos que contiene para organizar el código de la manera más sencilla y clara posible. Por otro lado, no debemos perder de vista que tenemos que intentar que nuestro código sea reutilizable y adaptable.

Por reutilizable, entendemos que si en un futuro se necesita hacer un proyecto nuevo que comparte algunos de los módulos que contiene este proyecto *Functionality*, el programador del nuevo proyecto debe poder utilizar el código del módulo que le interesa, sin necesidad de modificar el código de dicho módulo.
Además el código de este proyecto debe de ser adaptable, esto significa que si en un futuro se quieren añadir nuevos módulos o añadir nuevas características a los módulos ya existentes, el programador debe poder añadir dichos módulos o características sin necesidad de re-programar muchas de las líneas que ya contiene el proyecto.

Supongamos el siguiente escenario, tenemos a un usuario que posee un dispositivo móvil con acceso Wifi. En dicho dispositivo tiene instalado el programa que pretende desarrollar este proyecto y el usuario pretende usarlo para visualizar un canal de IPTV que se encuentra alojado en una determinada red.

El usuario lanza la aplicación y debe introducir por la interfaz, o GUI, los parámetros necesarios para la reproducción de su canal de IPTV. Dentro de estos datos, deberá especificar el servicio que desea lanzar, el canal que desea reproducir y el nombre del dominio que posee el servicio requerido, este dominio será la red IMS. A parte de esto parece lógico pensar que el usuario necesitará autenticarse en el servicio por lo que deberá proporcionar su nombre de usuario y su contraseña.

Cuando el usuario haya introducidos los datos y se inicie la ejecución del programa, este deberá verificar y validar los parámetros introducidos, asegurándose que el usuario ha escrito correctamente todos los datos necesarios para la ejecución. Para más información sobre este punto, recomiendo la lectura de la sección 6.1, donde se detallan los procedimientos de validación y verificación de los parámetros introducidos en el GUI.

Una vez validados todos los parámetros, el programa debe establecer una comunicación con la red IMS que posee el servicio solicitado. A través de dicha comunicación se procederá al establecimiento de la sesión utilizando el protocolo SIP.

Finalmente, cuando la sesión multimedia ha sido establecida el programa del usuario debe lanzar el reproductor multimedia para que reproduzca la sesión multimedia negociada anteriormente.

Analizando este escenario, podemos agrupar toda la funcionalidad en diferentes módulos. La agrupación llevaba a cabo en el desarrollo de esta práctica es la siguiente:

- **Interfaz o GUI**: Este módulo no entrará dentro de la funcionalidad del programa, como ya se ha explicado anteriormente, para dotar de mayor flexibilidad al programa y poder adaptarlo a diferentes dispositivos.

- **Mensajería**: Cuando el programa dialogue con el servidor el establecimiento de sesión, lo hará en un protocolo concreto, en este caso es SIP. Por lo que nuestra aplicación necesitará un módulo de mensajería que gestione los protocolos que se podrán utilizar en la comunicación con el servidor.

- **Conectividad**: Como nuestra aplicación tiene que establecer un diálogo con el servidor, necesitaremos de un módulo que gestione la conexión de nuestra aplicación con dicho servidor.

- **Utilidades**: Necesitaremos también un módulo de utilidades que contenga aquellos aspectos que sean propios de la ejecución del programa, que sean independientes de la arquitectura del dispositivo y que no se puedan catalogar dentro de los módulos anteriores. Dentro de este módulo estará la funcionalidad de ejecutar un programa externo, necesario para la ejecución del reproductor, y algunas funciones de autenticación de usuarios.

El proyecto *Functionality*, que contendrá la funcionalidad del cliente de IPTV, estará compuesto de los módulos de conectividad, mensajería y utilidades, como muestra la figura 3.2. Dichos módulos se describirán a continuación.

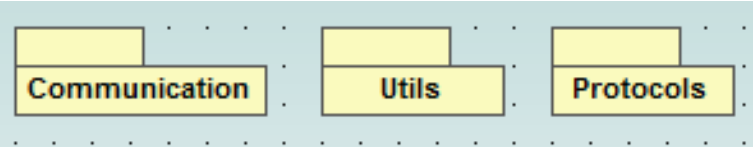

Figura 3.1: Composición del proyecto *Functionality*.

3.2.1. Módulo de mensajería

Como ya se ha explicado anteriormente, este módulo pretende dar soporte a todos aquellos protocolos que se utilicen durante la comunicación con el servidor. De momento, la comunicación con el servidor utiliza el protocolo SIP, por lo que dentro de dicho módulo tendremos un componente con toda la funcionalidad necesaria para el establecimiento y gestión de una conversación en SIP.

Para organizar el código de este módulo de una manera clara y sencilla, se han creado dos componentes dentro del módulo de mensajería. La descripción de estos componentes se detalla a continuación:

- **Messages**: En este componente se almacenan todas las clases que modelizan un mensaje concreto del protocolo SIP. Ejemplos de ficheros de esta carpeta son: *InviteMessage.cs*, *RegisterMessage.cs* o *AckMessage.cs*.

- **Utils**: El componente *Utils* nos permite agrupar aquellas clases que son necesarias para la creación de los diferentes mensajes almacenados en la carpeta *Messages*. Dentro de este componente aparecen los conceptos de SDP, representado por el fichero *SDP.cs*; Start Line o primera línea de todo mensaje SIP, representado en *StartLine.cs*; o las cabeceras, en *Header.cs*.

Una clase de vital importancia en el proyecto es la clase *MessageHandler.cs* que se encuentra dentro del módulo de mensajería SIP. Esta clase es tan importante, porque sirve de unión entre este módulo y el resto de las clases del proyecto. Las tareas de esta clase son las siguientes:

- **Creación de los diferentes mensajes SIP**: La tarea de creación de mensajes SIP es llevada a cabo por esta clase. Dependiendo del tipo de mensaje que se quiera crear, esta clase necesitará unos parámetros u otros; y con ellos, creará la totalidad del mensaje.

- **Gestión de las respuestas**: Una vez que se recibe una respuesta del servidor, esta clase puede acceder a cierta información de dicha respuesta. Puede comprobar que la respuesta se corresponde con la conversación que se está manteniendo en ese momento y puede acceder a diferentes parámetros del mensaje dependiendo del tipo de respuesta que sea.

- **Ocultación de la estructura del módulo**: Tal vez la característica más importante de esta clase, ya que oculta la estructura y organización del módulo de mensajería del resto de clases del proyecto. Esta clase es la única que conoce la presencia de las clases agrupadas en las carpetas explicadas anteriormente y oculta dicha organización con los tipos de datos que utiliza. Cuando una clase externa a este módulo quiere crear un mensaje SIP llama al método *CreateX*, donde 'Xés el método del mensaje que quiere crear: *Register*, *Invite*, etc. Todos estos métodos de creación de un determinado mensaje devuelven dos *String*, uno de ellos es el código de secuencia que tiene el mensaje creado y el otro, es el mensaje. Para la creación del mensaje SIP, esta clase sí trata los objetos de las clases almacenadas en SIP, pero cualquier otra clase externa sólo ve que la llamada al método *CreateRegister*, por ejemplo, devuelve dos *String* y no sabe nada de otras clases más específicas. Este sistema se puede observar en la figura 3.2.

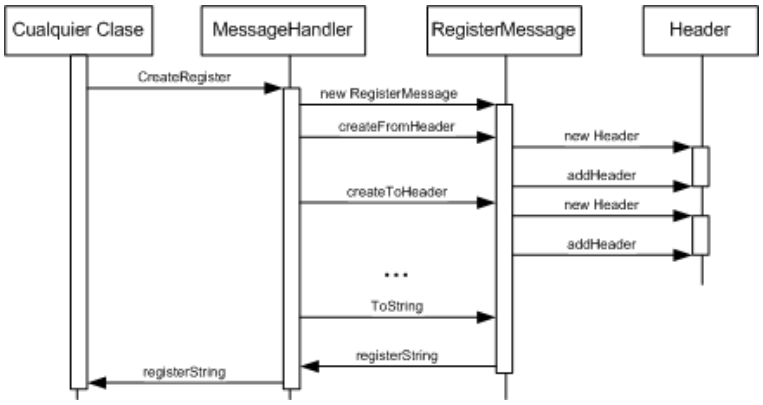

Figura 3.2: Diagrama de secuencia de la creación de un mensaje *Register*.

Este detalle dota de gran adaptabilidad a nuestro código, ya que si se quieren añadir nuevos mensajes al módulo de SIP sólo hay que modelar las nuevas clases que representen los mensajes y añadir a la clase *MessageHan-*

dler.cs los métodos que permitan manejar dichas clases.

Dentro de una conversación SIP existen gran variedad de mensajes, ver [15], pero muchos de ellos no son necesarios en el escenario en el que se desarrolla nuestro proyecto. Para conocer los mensajes que se deben modelar he realizado un estudio de los mensajes que se intercambiará el cliente con el servidor. En el escenario descrito anteriormente, durante la fase de establecimiento de la sesión multimedia, el programa intercambiará con el servidor los mensajes que se muestran en la figura 3.3.

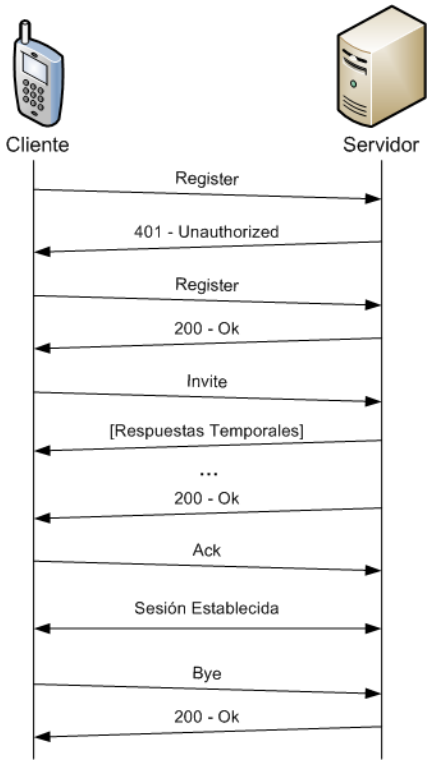

Figura 3.3: Conversación con el servidor de IMS.

Como se puede observar en la imagen 3.3, los mensajes que necesitamos son *Invite*, *Register*, *Ack* y *Bye*, que están modelados en las clases *InviteMessage*, *RegisterMessage*, *AckMessage* y *ByeMessage*, respectivamente.
Todas estas clases tienen la misma estructura y se componen de lo siguiente:

- **StartLine**: Es la cabecera con la que empiezan todos los mensajes SIP. Dependiendo de su formato nos indicará si el mensaje es una petición o una respuesta. Si es una petición, esta línea contendrá el tipo del

mensaje, la uri de la red de destino, y la versión de SIP utilizada. En cambio, si el mensaje es una respuesta contendrá la versión de SIP, el código de la respuesta y una descripción del mensaje. Decir que el código de las respuesta es un código de tres dígitos similar al utilizado en HTTP.

- **Header**: Son todas las cabeceras que tiene el mensaje SIP. Este objeto, para simplificar su uso, es un agregado de líneas de cabecera, modeladas en *HeaderLine.cs*, cuyo formato es *Atributo: Valor [;Nombre de la opción=valor; ...]*.

La clase *InviteMessage* contiene un atributo **body**, que sirve para poder añadir descriptores SDP al mensaje. El resto de clases no poseen este atributo ya que no lo necesitan. El contenido del componente *Messages* se puede ver en la figura 3.4

Por último decir que se ha modelado la clase *Response*, que sirve para obtener información de las respuestas que envía el servidor. Dicha clase posee métodos que permiten la creación de la respuesta a partir de un String, y una vez creada la respuesta podremos acceder a la información contenida en ella.

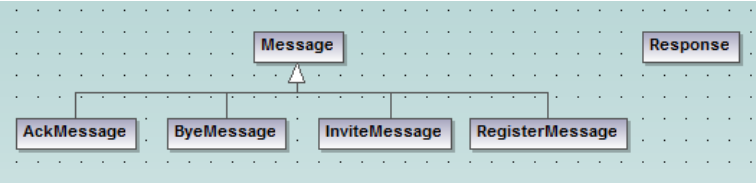

Figura 3.4: Contenido del componente Messages del módulo de mensajería SIP.

Como ya se ha explicado anteriormente, el componente *Utils* contiene clases auxiliares necesarias para la creación y gestión de los mensajes ya descritos. No se explicarán estas clases pues se tratan de clases de apoyo que tienen poco interés para ser mencionadas aquí.

3.2.2. Módulo de conectividad

El objetivo de este módulo es gestionar la comunicación con el servidor para poder establecer una conversación con él. Es el módulo más sencillo,

desde el punto de vista de clases involucradas, ya que este módulo está formado enteramente por la clase *ConnectionController.cs*.
Entre los atributos más importantes de esta clase se encuentran los siguientes:

- **Socket**: Este es el socket que nos permitirá enviar mensajes al servidor y recibir sus respuestas.
- **ClientPort**: Este es el puerto del cliente a través del cual se enviarán los mensajes y el puerto en el que se recibirán las respuestas.
- **Destination**: Este atributo es la dirección IP del servidor, será utilizado en el momento de conectar el socket.
- **ServerPort**: Al igual que el atributo *clientPort*, este atributo representa el puerto en el cual el servidor está a la espera de establecer la conversación.
- **ReceiveBuffer**: El atributo *receiveBuffer* es un array de bytes donde se almacenará el mensaje que se ha recibido del servidor.

Entre los métodos de esta clase merece la pena destacar los siguientes:

- **Send**: Con este método podremos enviar un mensaje a través del socket, al destinatario identificado por los atributos *destination* y *serverPort*.
- **Receive**: Nos permite recibir un mensaje a través del socket que de esta clase.
- **GetMessageReceived**: Permite obtener el mensaje recibido en un formato legible. El mensaje devuelto por este método es un String.

Con estos atributos y estos métodos, ya se puede establecer la comunicación con el servidor pero el problema surge cuando se envía un mensaje que genere varias contestaciones, ya que tal y como se programó inicialmente esta clase, el método *receive* finalizaba cuando conseguía la primera contestación. Para solventar esta situación se tomó en consideración la creación de un bucle. Si el problema era que existían respuestas temporales, la solución podría consistir en crear un bucle dentro del método *receive* que permitiese al método esperar hasta que tuviese una respuesta definitiva. Esta solución fue rechazada porque mientras el programa estuviese en el bucle esperando la respuesta definitiva, se perdería el control de la aplicación, por lo que en la interfaz no se podrían mostrar los mensajes correspondientes a las respuestas temporales enviadas por el servidor.

Por lo tanto, la solución requería que se recibiesen todos los mensajes y mientras estos llegaban, se fuera notificando al usuario de la información contenida en esos mensajes intermedios. La solución ha sido programar esta clase con hilos o *threads*.
El thread entra en ejecución cuando esperamos recibir alguna contestación del servidor. Este thread, lo que hace es recibir todas las contestaciones del servidor hasta que le llega una contestación definitiva. Como se ha explicado en el Módulo de mensajería, sección 3.2.1, nuestra aplicación utilizará el protocolo SIP y por tanto todas las respuestas definitivas tienen un código mayor o igual a 200.
Todas las respuestas intermedias y la última respuesta definitiva son almacenadas en un vector de mensajes, lo que permite que el programa acceda a los mensajes recibidos, los procese y muestre la información por pantalla. Por lo que la inclusión de este thread, a obligado a la utilización de nuevas variables necesarias para la sincronización y gestión de los recursos compartidos entre el *thread* y el programa padre.

3.2.3. Módulo de utilidades

El tercer módulo de este proyecto es el módulo de utilidades el cual contiene algunas utilidades que sean necesarias para la correcta ejecución del programa y que no puedan ser catalogadas dentro de los módulos anteriores. Dentro de este módulo encontramos las clases: *Hash*, *ExternalProgram* y *RandomString*.

Clase Hash

La clase *Hash* sirve para calcular el hash MD5 de una cadena de caracteres específica. Esta clase será utilizada a la hora de iniciar sesión y validar al usuario. Como ya se vio en la sección 2.2.5, la manera de autenticar a un usuario en el sistema es utilizando el método *HTTP Digest* o *Resumen HTTP*. Sin embargo, como también se vio en la sección anteriormente referenciada, el cálculo de la *hash* puede realizarse con diferentes algoritmos como pueden ser MD5 o SHA1; pero para nuestro proyecto utilizamos únicamente el algoritmo MD5 debido a que el core instalado en el servidor sólo soporta este algoritmo. El procedimiento de validación es el siguiente:

1. El servidor envía un reto al usuario. Dicho reto es el valor de la opción *realm*, dentro de la cabecera *WWW-Authenticate* de la respuesta *401 - Unauthorized*. En esta cabecera también se especifica el método de

cifrado y se envía un *nonce*, que es una cadena que envía el servidor para añadir frescura a la clave y evitar ataques de terceras personas.

2. El programa verifica la existencia de la cabecera *WWW-Authenticate* y comprueba que el método de cifrado es el soportado por la aplicación.

3. Si las comprobaciones anteriores son correctas, el programa obtiene el realm del mensaje del servidor y calcula un primer hash, llamémosle *hash1* formado por:

   nombreDeUsuario : realm : passwordDelUsuario

4. Seguidamente, se calcula otro hash, llamémosle *hash2*, que es el resultado de cifrar:

   métodoDelMensaje : uri

   El método del mensaje será REGISTER y la uri es la dirección de la red a la que nos queremos conectar.

5. El contenido de *hash2* se pasa a hexadecimal y conseguimos los bytes de esa cadena en hexadecimal pero codificados utilizando *UTF-8*. Al resultado lo vamos a llamar *resultadoHash2*.

6. El siguiente paso consiste en conseguir un tercer hash, *hash3*, que es el resultado de aplicar el algoritmo MD5 sobre la cadena formada por:

   hash1 : nonce : resultadoHash2

7. Finalmente, la respuesta al reto inicial del servidor son los bytes del contenido en hexadecimal de *hash3* codificados con *UTF-8*.

Clase ExternalProgram

Una vez establecida la sesión con el servidor, el programa *IPTV* necesita que un reproductor se conecte por *rstp* a la dirección que indique el servidor. El objetivo de esta clase es, precisamente, ejecutar el reproductor multimedia y hacer que se conecte a dicha dirección. Para ello, la clase posee tres atributos que son los siguientes:

- **Name**: Es el nombre del fichero de ejecución del reproductor.

- **Path**: Es la ruta absoluta donde se encuentra el fichero descrito por la variable *Name*.
- **Arguments**: Son los argumentos necesarios para que el programa se conecte a la dirección indicada por el servidor.

Todos estos atributos tendrán un valor diferente dependiendo del proyecto que estemos ejecutando, ya que no existen los mismos reproductores para ordenador que para dispositivos móviles, y además, las rutas tampoco son iguales. Por estas razones, el tratamiento de los nombres de los ficheros, sus rutas y sus parámetros deben ser tratados por cada proyecto.

Clase RandomString

Como su propio nombre indica, esta clase genera cadenas de caracteres de forma aleatoria. La clase posee un único método que se llama **randomNumber** cuya labor es generar un número aleatorio de 'X'dígitos. Los argumentos de este método son el número de dígitos que queremos que tenga el resultado y una semilla que transmita aleatoriedad al resultado. Esta clase y este método son utilizados para rellenar el campo *branch* de los mensajes SIP que sirve para identificar las transacciones de mensajes.

3.3. Descripción de los proyectos IPTV

En la sección anterior se ha explicado el proyecto *Functionality*, el cual es un *Class Library* que contiene la funcionalidad de la aplicación. Las razones por las que se creó el proyecto de este tipo, es para poder crear diferentes proyectos para los diferentes dispositivos. Estos proyectos al importar o utilizar la *.dll* del proyecto *Functionlity*, tendrán toda la funcionalidad del cliente de IPTV.
En esta sección se describirá el proceso de desarrollo de los proyectos *IPTV-Client* y *IPTVClient-PC*. El primer proyecto, es utilizado para dispositivos móviles; mientras que el segundo permite ejecutar el programa en ordenadores personales. Ambos proyectos poseen la misma estructura y contienen prácticamente la misma información y el mismo código, por tanto, no es necesario especificar el desarrollo del proyecto *IPTVClient* y después hablar de *IPTVClient-PC*, ya que la única diferencia serán los nombre de los reproductores y sus rutas.
Como las diferencias entre estos dos proyectos son mínimas se planteó la posibilidad de colocar el código común, el fichero *Core.cs*, dentro del proyecto *Functionality*. Pero esto no es posible ya que la clase *Core* contenida en dicho

fichero, entre otras tareas maneja la interfaz y es la encargada de imprimir los mensajes en ella. Al colocar dicha clase en el proyecto anteriormente descrito, el programa no podría saber a qué interfaz hace referencia si a la del proyecto para ordenador o a la del proyecto para dispositivos móviles.
La estructura de estos proyectos es la misma:

- **GUI.cs**: Este fichero contiene la interfaz del programa y por tanto, debe pedir al usuario todos los datos necesarios para el correcto funcionamiento del proyecto *Functionality*.
- **MainFile.cs**: En este fichero se encuentra el método *main*, cuyo objetivo es lanzar la aplicación. Este fichero es el más simple de todos los proyectos y no se describirá ya que únicamente contiene ese método *main* que lanza la interfaz.
- **Core.cs**: Si debo seleccionar una clase y catalogarla como la clase vital para el proyecto, dicha clase es la clase *Core* contenida en este fichero. Esta clase es la encargada de unir el proyecto *Functionality* con la interfaz.

Ahora se explicarán más en detalle el contenido y funciones de los ficheros *GUI.cs* y *Core.cs*.

3.3.1. GUI.cs

Este fichero contiene la interfaz de la aplicación que debe pedir todos los datos necesarios para el inicio de la sesión. No debemos perder de vista que una buena interfaz debe de ser clara y fácil de usar, para que el usuario sepa y se sienta cómodo a la hora de usarla.
Analizando los mensajes SIP que se deben intercambiar, necesitaremos de los siguientes datos:

1. **Usuario o Dirección Pública**: Esta dirección es asignada por el operador al usuario. Tiene la forma de una una URI SIP, *nombreDeUsuario@operador*, y sirve para encaminar los mensajes del servidor y la sesión. Es una dirección de contacto. Para más información sobre este dato se recomienda la lectura de la sección 2.2.4.
2. **Dirección Privada**: Esta dirección no es una URI SIP, sino un identificador de acceso a la red o NAI. Esta dirección será utilizada para identificar subscripciones y para realizar la autenticación del usuario. Para más información sobre este dato se recomienda la lectura de la sección 2.2.4.

3. **Contraseña**: Es la contraseña establecida por el operador al usuario. También será necesaria en el proceso de autenticación.

4. **Red**: Este campo identifica a la red de destino, red que nos proporcionará el servicio que queremos iniciar.

5. **Servicio**: El servicio es el recurso que queremos iniciar y que está en la red IMS identificada por el campo Red.

6. **Canal**: Al tratar este proyecto sobre un servicio de IPTV, deberemos seleccionar el canal de televisión que queremos visualizar.

Como se puede deducir, los campos que acabamos de describir pueden ser agrupados en dos pestañas que serán:

- **Pestaña de Usuario**: Esta pestaña contendrá los datos referentes al usuario. Estos datos son el nombre de usuario, la dirección privada y su contraseña. Hay que destacar que si esta aplicación se lleva a un dispositivo SIP, los campos del nombre de usuario y su dirección privada no se pedirían en la interfaz. Esto se debe a que los dispositivos SIP utilizan una tarjeta UICC, Universal Integrated Circuit Card. Dicha tarjeta contiene una aplicación ISIM, IP multimedia Services Identity Module, que sirve para identificar al usuario en los servicios multimedia IP. El contenido de esta aplicación se puede ver en la figura 3.5. Por lo tanto, como el nombre del usuario y la dirección privada ya las proporciona la tarjeta ISIM, la aplicación deberá acceder a la información contenida en la tarjeta en lugar de pedírsela al usuario.

- **Pestaña de Red**: La información que sirve para identificar la red en la cual queremos iniciar el servicio, así como el nombre del servicio y el canal, estarán ubicados en la Pestaña de Red. Esta información es necesaria y debe introducirla el usuario.

Aparte de las pestañas descritas anteriormente, se han creado otras dos pestañas que son las siguientes:

- **Pestaña de Opciones**: En esta pestaña se permite al usuario personalizar la ejecución de la aplicación. El usuario puede elegir si desea o no, un fichero de detalles o *Log*. En el caso de que sí desee la creación, podrá seleccionar tanto el nombre como la ubicación del fichero. Además, puede elegir con qué aplicación desea reproducir el servicio multimedia que ha especificado

Figura 3.5: Contenido de una aplicación ISIM.

- **Pestaña de Sesión**: Finalmente, en esta pestaña el usuario podrá iniciar sesión y se mostrarán mensajes que describan el estado de la sesión iniciada.

Aparte de contener el diseño de la interfaz, esta clase es la encargada verificar el contenido de los campos necesarios para la ejecución y de iniciar la sesión con los datos introducidos por el usuario. Si se desea ampliar los conocimientos sobre las técnicas de validación que realiza la aplicación, se recomienda la lectura del capítulo 6. Si por el contrario se desea ver la apariencia de las interfaces así como conocer la manera en la cual ejecutar la aplicación, por favor, pase a la apéndice B.

3.3.2. Core.cs

La clase *Core* se encuentra contenida en el fichero *Core.cs*. Esta clase es de vital importancia para el proyecto, ya que es la encargada de utilizar los datos que ha introducido el usuario por pantalla junto con los procedimientos definidos en el proyecto *Functionality*.
Lo primero que hace esta clase, es comprobar que el dispositivo dispone de una dirección IPv4. Las razones por las que hace esto es porque el servidor sobre el que probaremos el programa no trabaja con direcciones IPv6.
El método principal de esta clase se llama *startProcess*. Este proceso se ejecuta cuando el usuario hace click sobre el botón Íniciar Sesión'de la interfaz. Lo primero que hace el programa es verificar que el dispositivo está conectado a Internet. Para ello, utiliza el método *connected* de la clase *ConnectionController* el cual hace una petición HTTP a una página web.
Si el dispositivo se encuentra conectado a Internet, empieza la fase de aut-

enticación. Esta fase se encuentra programada en el método *authenticate* y se compone del envío del primer mensaje *REGISTER*, analiza la respuesta y, si es necesario, calcula la respuesta al reto enviado por el servidor para autenticar al usuario.
Cuando termina la fase de autenticación, el método *startProcess* analiza el resultado de dicha fase. En caso de que se haya producido algún error, se informa al usuario a través de la interfaz y del fichero de detalles, siempre que este haya sido establecido. En caso de que el usuario se haya autenticado correctamente, este método pasará a la fase de inicio de sesión.
Esta fase comienza con el envío del mensaje *INVITE*, y va actualizando la interfaz con los mensajes temporales enviados por el servidor. Al igual que en la fase anterior, cualquier error es notificado al usuario a través de la interfaz y del fichero de error, siempre que corresponda. Finalmente si este proceso finaliza con un resultado satisfactorio, se obtiene, del mensaje del servidor, la dirección a la que se debe conectar el reproductor y se ejecuta dicho reproductor para acceder al recurso.

3.4. Modelo de programación aplicado al proyecto

Una vez que se ha descrito el proceso de desarrollo de la aplicación, en esta sección se describirá el patrón de programación utilizado este proyecto. Además de presentar el patrón, se identificarán los componentes de dicho patrón dentro de nuestra aplicación.

3.4.1. Modelo Vista Controlador

El patrón utilizado es el patrón arquitectónico *Modelo Vista Controlador* o *MVC*, de su nombre en inglés *Model View Controller* [43].
Este patrón se caracteriza por separar la lógica de la aplicación, llamada Modelo; la interfaz de la aplicación, llamada Vista; y los datos de la aplicación llamado Controlador. La figura 3.6 muestra el diseño de este patrón.

El funcionamiento de este patrón es el siguiente:

1. El usuario realiza alguna acción sobre la Vista que provoca que se ejecute un procedimiento dentro del Controlador.

2. El Controlador maneja la información recibida por la Vista y actualiza el estado del Modelo.

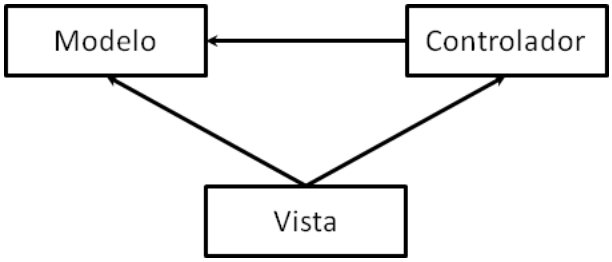

Figura 3.6: Diseño del patrón Modelo-Vista-Controlador.

3. El Modelo delega en la Vista las tareas de actualización de la interfaz. Dicha actualización dependerá de la acción efectuada por el usuario en el paso 1.

3.4.2. Identificación en el proyecto

Una vez que se tiene un conocimiento general de este patrón, llega el momento de identificar los diferentes componentes de nuestro proyecto que actúan como Vista, Controlador y Modelo.

Es obvio una primera asociación entre Vista y la clase *GUI.cs* ya que esta clase es la encargada de representar la interfaz de usuario y captar todas las acciones que este realiza.
Cuando el usuario pincha sobre el botón *Iniciar Sesión*, los datos que ha introducido el usuario en la interfaz son enviados a la clase *Core.cs*, la cual realiza las funciones del Controlador. Esta clase será la encargada de validar los datos de la interfaz y de "actualizar" el estado del Modelo.
El Modelo, en nuestro proyecto, se corresponde con el proyecto *Functionality* el cual contiene toda la funcionalidad de la aplicación. Hay que matizar que, en nuestro proyecto, el Modelo no se actualiza con cada iteración del patrón sino que comienza la negociación del establecimiento de la sesión. El resultado de dicha negociación será representado en la Vista a través del Controlador, ya que el resultado es recibido por la clase *Core.cs* y en función de la respuesta, esta clase mostrará un mensaje u otro en la interfaz. El diseño del patrón permite ligeras modificaciones en el funcionamiento del mismo y, si tenemos en mente la idea de independencia de los módulos, no parece razonable que el Modelo tenga conocimiento de la Vista.

Capítulo 4

Desarrollo de un reproductor multimedia para Windows CE

En este capítulo se va a describir el proceso de desarrollo del reproductor multimedia para dispositivos móviles basados en Windows CE. Una vez que se ha desarrollado el cliente de IPTV, cuyos detalles están descritos en el capítulo 3, el siguiente paso es conseguir un reproductor que reproduzca el servicio que se ha especificado en la interfaz.

4.1. Requisitos que debe cumplir el reproductor

En esta sección se describirán los requisitos que debe cumplir el reproductor multimedia que utilizaremos en el proyecto. Ya se ha dicho en numerosas ocasiones, a lo largo de esta memoria, que se debe encontrar un reproductor que satisfaga nuestras necesidades, pero no se han especificado esas necesidades.
El requisito que debe cumplir el reproductor es que debe de ser de código abierto. Ya que de esta manera, se podrá modificar el código y añadirle el módulo de acceso a los contenidos de pago o PPV. Este es requisito fundamental si queremos añadir el sistema de PPV. También podemos omitir este requisito para probar la correcta ejecución de la aplicación, por lo que los requisitos que debe cumplir el reproductor son:

- **Disponible para Windows CE**: Obviamente, el reproductor deberá estar disponible para la plataforma objetivo que es el dispositivo móvil basado en Windows CE.

- **Soporte para el tipo de conexión admitida por el servidor**: Cuando termina la ejecución del cliente SIP, el servidor envía una url en la cual se encuentra el servicio. Esta url está basada en un determinado protocolo de streaming que deberá ser soportado por el reproductor para poder acceder al servicio y reproducirlo.

El objetivo principal será buscar un reproductor que nos permita añadir el módulo de PPV. Sin embargo, no se perderá de vista la búsqueda de otro reproductor que sea capaz de reproducir el servicio solicitado, a pesar de no tener soporte para PPV. Para este último caso, necesitaremos un reproductor que soporte conexiones rtsp. Esta búsqueda tendrá como objetivo la validación del funcionamiento de la aplicación en dispositivos móviles.

4.2. Proceso de migración del reproductor VideoLan Client

Una vez que han sido analizados los diferentes reproductores multimedia en la sección 2.4, se debe seleccionar el reproductor que utilizará nuestra aplicación. El reproductor seleccionado para esta tarea es VLC, cuyas características están descritas en la sección 2.4.4. Las razones por las que se ha elegido este reproductor son las siguientes:

1. Al ser un proyecto de código abierto, podremos añadir al código el módulo de PPV necesario para proporcionar la protección de video deseada.

2. Otra razón es la cantidad de formatos que soporta el reproductor, los cuales proporcionan flexibilidad a nuestro programa al no depender, en gran medida, del formato que reproduce el servidor.

Previamente a la inclusión del módulo de PPV al código de reproductor VLC se intentó obtener el reproductor en su versión para Windows CE para probar la funcionalidad de la aplicación. Según la página de VideoLan, este programa no se encuentra portado a la plataforma que nos interesa, por tanto es tarea nuestra el portar el código de manera manual a la plataforma de Windows CE.

Una vez descargado el código y descomprimido, se observa un fichero llamado "INSTALL.wince" el cual se puede seguir para instalar el programa en nuestro dispositivo. Al seguir la información contenida en ese fichero no

pude compilar el programa ya que obtuve numerosos mensajes de error.

Para solventar esto, publiqué un mensaje con mi problema y me dijeron que el problema es que no se utiliza el cross-compiler que se recomienda en ese fichero, el cual es "arm-wince-pe". En su lugar, me recomendaron la guía del tutorial publicado en [44].

Una vez leído dicho tutorial, consigues tener una idea general del proceso al que te estás enfrentando. El proceso de migración se compone de tres fases:

1. **Proceso de arranque o bootstrap**: Esta es la primera fase y con ayuda de un comando, se generan los ficheros necesarios para la compilación del código. En esta fase, se especifica el cross-compiler que se va a utilizar, la ubicación de dicho cross-compiler, y algunos parámetros que determinarán el proceso de compilación del código como características que deberemos habilitar o deshabilitar.

2. **Proceso de compilación**: Una vez que el código la fase anterior ha finalizado correctamente, se debe compilar el código. Para ello, se deberá ejecutar el programa "make".

3. **Proceso de generación del instalable**: Finalmente, cuando el programa ya está compilado, es necesario crear el instalable para la plataforma, para ello, basta con ejecutar la siguiente sentencia en el terminal:

   make package-win32-base

Por lo tanto, una vez que conocemos esa información, me dispuse a la realización de la primera tarea. Para ello, partiendo del ejemplo que aparece en el wiki anteriormente mencionada:

```
PATH=/opt/mingw32ce/bin:/opt/mingw32ce/arm-wince-mingw32ce/bin:$PATH\
CC="arm-wince-mingw32ce-gcc" \
CXX="arm-wince-mingw32ce-g++" \
CFLAGS="-I/usr/wince/include
 -I/opt/mingw32ce/arm-wince-mingw32ce/include
   -mwin32 -D __COREDLL__ -D _WIN32_WCE=0x0500" \
CPPFLAGS="-I/usr/wince/include
 -I/opt/mingw32ce/arm-wince-mingw32ce/include  -mwin32
 -D __COREDLL__ -D _WIN32_WCE=0x0500" \
LDFLAGS="-L/opt/mingw32ce/arm-wince-mingw32ce/lib -L/usr/wince/lib"\
```

```
./configure --host=arm-wince-mingw32ce \
            --disable-sdl --disable-gtk \
            --disable-dvdnav --disable-dvdread --disable-avformat \
            --disable-postproc --disable-hal --disable-nls    \
            --enable-sout --enable-vlm --disable-wxwindows \
            --disable-a52 --enable-libmpeg2 --disable-freetype \
            --disable-libgcrypt --disable-fribidi --disable-mad \
            --disable-optimize-memory --disable-audioscrobbler \
            --disable-tremor --disable-faad --enable-ffmpeg \
            --disable-avcodec --enable-vlc --disable-activex \
            --disable-testsuite --disable-skins2 \
            --disable-qt4 --disable-notify --disable-httpd \
            --disable-dbus-control --disable-growl \
            --disable-telepathy --disable-lua --disable-vlm \
            --disable-gnutls --disable-bonjour --disable-x11 \
            --disable-glx --disable-xvideo --disable-remoteosd \
            --disable-schroedinger --disable-dshow \
            --enable-wingdi --disable-real --disable-realrtsp \
            --disable-optimizations --enable-debug \
            --enable-wince --enable-waveout --disable-directx \
            --disable-x264 --disable-live555 --disable-pulse \
            --disable-swscale --disable-telnet
```

Y teniendo en cuenta todas las opciones que se pueden utilizar con el comando *./configure*, inicié el proceso de arranque. Para finalizar esta tarea de forma satisfactoria hizo falta la instalación de numerosos paquetes como *autoconf*, *libtool*, *libgcrypt-dev*, *gettext*, y *cvs*.

Mientras solventaba algunos de los errores que aparecieron durante la fase de arranque, me recomendaron el uso del canal de IRC de videolan, para poder obtener dudas más rápidamente sin depender del foro. Allí, me informaron que la última versión del código posee numerosos errores que dificultan el proceso de migración de la aplicación y me recomendaron el uso de versiones 0.9.x ya que parecen ser las más fiables. Además me informaron que este código no dispone de interfaz, por lo que deberíamos crearla antes de compilar el programa.

A pesar de esto, al intentar compilar el programa nos encontramos con el siguiente error:

LibVLC requires mingw-runtime

Este error seguramente se deba a que el cross compiler utilizado está basado en *cigwing* y el proceso de compilación requiere la utilización de otro cross compiler basado en *mingw*. Sin embargo, cuando se llegó a esa solución ya se había alcanzado la fase final del desarrollo del proyecto, lo que paralizaba el desarrollo en esta línea de investigación.

Capítulo 5

Historia del proyecto

Una vez que se ha explicado el desarrollo de los proyectos involucrados en este estudio, se describirá la planificación de este proyecto así como los problemas más importantes que han afectado al seguimiento de esa planificación. A parte de esto, se intentará estimar un presupuesto para este proyecto.

5.1. Planificación

Para poder afrontar con éxito el desarrollo de cualquier proyecto, es necesario realizar una planificación de las tareas que se verán involucradas en el desarrollo. El objetivo de esta sección será explicar la planificación que se ha llevado a cabo durante el desarrollo de este proyecto.

En la tabla 5.1 se muestra el nombre, la duración, la fecha de comienzo y la fecha de fin de las diferentes tareas llevadas a cabo.

A continuación, se describirán los grupos de tareas que aparecen en la tabla 5.1 y también se explicarán algunos detalles que aparecen en dicha planificación.

5.1.1. Estudio Inicial

En este grupo de tareas entran todo el estudio realizado antes del desarrollo del proyecto. Como se puede observar en la tabla, el estudio se realizó sobre la arquitectura IMS y sobre el protocolo SIP. Pero no fue lo único, ya que se estudiaron otros temas, como por ejemplo, los SDP o los tipos de reproductores existentes para los dispositivos móviles. Estos

Nombre de tarea	**Días**	**Inicio**	**Fin**
Estudio Inicial	**24**	**mar 10/02/09**	**vie 13/03/09**
Estudio sobre IMS	14	mar 10/02/09	vie 27/02/09
Estudio sobre SIP	10	lun 02/03/09	vie 13/03/09
Análisis y Diseño de la aplicación	**10**	**lun 16/03/09**	**vie 27/03/09**
Análisis de las necesidades	3	lun 16/03/09	mié 18/03/09
Diseño detallado	7	jue 19/03/09	vie 27/03/09
Desarrollo	**50**	**lun 13/04/09**	**vie 19/06/09**
Desarrollo del proyecto *Functionality*	35	lun 13/04/09	vie 29/05/09
Desarrollo del proyecto *IPTVClient-PC*	5	lun 01/06/09	vie 05/06/09
Desarrollo del proyecto *IPTVClient*	10	lun 08/06/09	vie 19/06/09
Pruebas	**10**	**lun 22/06/09**	**vie 03/07/09**
Prueba del proyecto *IPTVClient-PC*	7	lun 22/06/09	mar 30/06/09
Prueba del proyecto *IPTVClient*	3	mié 01/07/09	vie 03/07/09
Desarrollo del reproductor	**43**	**mié 01/07/09**	**vie 28/08/09**
Compilación de *live555*	23	mié 01/07/09	vie 31/07/09
Compilación de *VLC*	20	lun 03/08/09	vie 28/08/09
Documentación	**40**	**lun 06/07/09**	**vie 28/08/09**
Documentación del código	3	lun 06/07/09	mié 08/07/09
Desarrollo de la memoria	37	jue 09/07/09	vie 28/08/09
Revisión del código y de la memoria	12	lun 31/08/09	mar 15/09/09

Cuadro 5.1: Planificación del proyecto.

estudios no se han añadido a la planificación ya que no son tan importantes como la arquitectura IMS y el protocolo SIP.

Estos dos últimos estudios son necesarios remarcarlos debido a la gran relevancia que tienen dentro del proyecto. Además, mis estudios sobre Ingeniería Informática no cubrían estos temas, por lo que el nivel de estudio tuvo que ser mayor al que puede realizar cualquier estudiante de Ingeniería Informática.

5.1.2. Análisis y Diseño de la aplicación

Una vez que se dominan los conceptos importantes que utilizarán a lo largo del proyecto, comienza la fase de análisis y diseño y de la aplicación. En esta fase, se analizan las necesidades que va a tener la aplicación y se comienza a crear un diseño con los módulos o los componentes más importantes.

Respecto a las necesidades de la aplicación, esta fase se centró más en los datos que necesarios para la aplicación por parte del usuario, como el nombre de la red de destino o el nombre del servicio, que en las necesidades o características que necesitaría tener el dispositivo. Esto se debe a que necesidades tales como la memoria del dispositivo o la capacidad del procesador, no son importantes ya que nuestra aplicación se compone, básicamente, de un cliente SIP y de un reproductor multimedia. Por lo que no requiere mucha capacidad de procesamiento o mucha memoria.

En lo referente al diseño de la aplicación, cabe reseñar que esta tarea ha sido una de las que más se ha cambiado desde su primera versión. Inicialmente, la idea era realizar un único proyecto que fuera la aplicación para dispositivos móviles, pero con el paso del tiempo este diseño tuvo que ser cambiado hasta llegar al diseño final, el cual se compone de un proyecto llamado *Functionality* que contiene la funcionalidad del cliente, y otros dos proyectos que son la aplicación para ordenador, *IPTVClient-PC*, y la aplicación para dispositivos móviles, *IPTVClient*.

5.1.3. Desarrollo

El desarrollo del proyecto se compone de tres tareas las cuales son los tres proyectos que se presentan en este estudio. Esta fase es la fase más larga de todo el proyecto ya que contiene toda la programación de las aplicaciones desarrolladas.

Como se observa en la tabla, el desarrollo del proyecto *Functionality* es el más largo de las tres tareas. Esto es lógico debido a las siguientes razones:

1. Los otros dos proyectos contienen la interfaz de la aplicación y un subconjunto muy reducido de la funcionalidad que depende del proyecto en cuestión.

2. Como ya se ha explicado anteriormente en la sección 3.2, el proyecto *Functionality* se compone de varios módulos como son el módulo de mensajería, el módulo de conectividad y el módulo de utilidades.

3. Durante el desarrollo del proyecto *Functionality* tuvimos numerosos problemas en la ejecución de la aplicación. Estos problemas se debían a que los mensajes SIP estaban mal formados, en cuyo caso la solución era sencilla, pero el mayor problema encontrado en esta fase fue el método de autenticación del usuario en la red. Este problema se verá más en detalle, en la sección 5.2.

Por otro lado, el desarrollo de la aplicación para los diferentes dispositivos es algo relativamente sencillo, una vez que tenemos el proyecto anteriormente descrito. Como se ha explicado numerosas veces en esta memoria, para el desarrollo de la aplicación en un determinado dispositivo es necesario la creación la interfaz y la creación de una clase, llamada *Core*, que realice la unión entre la interfaz y los métodos existentes en el proyecto *Functionality*. A pesar de esto, como podemos observar en la tabla, el desarrollo de la aplicación para dispositivos móviles duró más tiempo que el desarrollo para ordenadores. Esto se debe a que a la hora de desarrollar el código de la clase *Core* surgieron problemas que alargaron el tiempo de desarrollo.

Un detalle que llama la atención es que entre el 27 de Marzo y el 13 de Abril no se desarrolló ninguna tarea. Esto se debe a que en ese período tuvo lugar el viaje fin de carrera que me impidió el desarrollo de cualquier tarea. Perfectamente se podría haber programado alguna, pero no se iba a realizar; por lo que la mejor planificación consistía en la reserva de esos días. Ese paréntesis, provocó un cierto período de productividad relativamente baja en los primeros días de la fase de desarrollo que alargaron la planificación de esta fase.

5.1.4. Pruebas

Finalmente, cuando ya se han desarrollado todos los proyectos es necesario lanzar una batería de pruebas para asegurarse del correcto funcionamiento

de la aplicación. Todo lo referente a las pruebas realizadas, está descrito en el capítulo 6.
La fase de pruebas del programa para ordenador duró más que las pruebas para dispositivos móviles, ya que en la primera tarea también se probó el correcto funcionamiento del proyecto *Functionality*. Cuando esta tarea finalizó, el proyecto que contiene la funcionalidad del programa estaba totalmente probado, por lo que las pruebas del proyecto *IPTVClient* sólo se debían centrar en el tratamiento de excepciones relacionadas con la interfaz.

5.1.5. Fase de documentación

Una vez terminada la fase de pruebas, comenzó la fase de documentación del proyecto. Esta fase de documentación se compone de las siguientes tareas:

- Comentar el código creado en los tres proyectos.
- Generación de esta memoria.

La tarea de comentar el código no duró mucho en comparación con la tarea de elaboración de esta memoria. Esto es lógico, ya que los contenidos que debe abarcar esta memoria son bastante extensos.

5.1.6. Presupuesto del proyecto

Una vez desarrollado el proyecto debo de proporcionar un presupuesto orientativo del coste de este proyecto. Para realizar esta tarea, creo que lo más conveniente es calcular el número de horas invertidas en el proyecto y multiplicarlas por el precio que puede tener una hora de trabajo.

Teniendo en cuenta esto, consideraré que, por termino medio, se han empleado 5 horas diarias en la realización de este proyecto. Este dato es una estimación y no es real, ha habido numerosos días en los que las horas han excedido esta cantidad con creces pero que compensan con otros días menos productivos.

Por otro lado, analizando la planificación de la tabla 5.1, vemos que se han invertido un total de 134 días en el desarrollo del proyecto. Este dato se ha obtenido sumando los días de las siguientes fases:
Para el cálculo de este presupuesto, no tendremos en cuenta la fase de desarrollo del reproductor ya que esta tarea se realizó conjuntamente con la

Fase	**Días**
Estudio Inicial	24 días
Análisis y Diseño de la aplicación	10 días
Desarrollo	50 días
Pruebas	10 días
Documentación	40 días
Total	**134 días**

fase de documentación. Y tampoco se tiene en cuenta los días de revisión ya que todo el proyecto está desarrollado al inicio de esta tarea.

Si consideramos un precio estimado de 15 €/h. Tenemos que el número de horas son 670 horas y el precio del desarrollo del proyecto asciende a 10050 €.

$$Precio = Dias * NumeroDeHoras * PrecioPorHora = 134 * 5 * 15 = 10050 \quad (5.1)$$

5.2. Problemas encontrados

En esta sección se describirán los problemas a los que nos hemos enfrentado durante el desarrollo del proyecto. A pesar de todos los problemas que se han producido, en esta sección sólo se detallarán los más importantes y los que han producido retrasos en la planificación del proyecto. Además de la descripción del problema se detallarán los medios utilizados para solucionar el problema, siempre y cuando dicho problema haya sido resuelto.

1. Acceso a internet con una IP privada
2. Uso de una dirección IPv6
3. El método de autenticación del usuario en la red IMS
4. Reproductor multimedia para Windows CE

A continuación se proporciona una descripción más detallada de cada uno de estos problemas.

5.2.1. IPs privadas

El problema por el uso de IPs privadas fue el primero que surgió. Una vez que se decidió el desarrollar el cliente para el ordenador, proyecto *IPTVClient-PC*, se comenzó el desarrollo de dicho proyecto desde fuera de la universidad. El problema que surgió fue que en dicho lugar, el ordenador estaba colocado detrás de un router y estaba identificado por una IP privada. Las repercusiones que tenía este hecho, se hacían evidentes a la hora de ejecutar el cliente, ya que el programa podía enviar los mensajes a la red IMS perfectamente pero la contestación de la red nunca llegaba al ordenador.

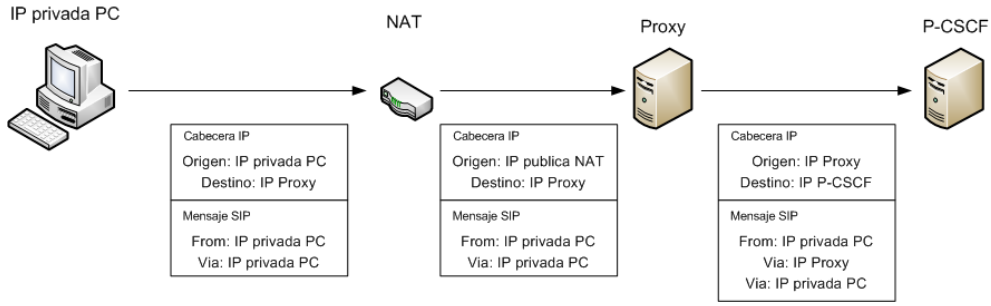

Figura 5.1: Envío de un mensaje SIP desde un dispositivo colocado tras una NAT.

La figura 5.2.1 representa el proceso de envío de un mensaje SIP desde un ordenador colocado detrás de una NAT. Como podemos observar el envío se realiza correctamente ya que el ordenador añade su IP privada al campo Origen de la cabecera IP y en el campo Via del mensaje SIP.
Este campo Via sirve para identificar el camino que ha seguido el mensaje, para que la contestación pase por los mismos nodos. Cada vez que un mensaje SIP pasa por un nodo, dicho nodo añade una cabecera SIP al mensaje, para que la contestación pase por él. Estas cabeceras Via se tratan como una pila, utilizando el método LIFO, lo que quiere decir que la última entrada añadida será la primera procesada.
La NAT lo que hace es asociar la IP privada del ordenador a un descriptor de sesión, y cambia la IP privada del campo Origen de la cabecera IP por su propia IP publica. Y aquí es donde está el problema ya que cuando el destinatario procese conteste a dicho mensaje, enviará dicho mensaje al nodo identificado en el campo Via. Y esto se hará así sucesivamente hasta que el mensaje llegue al proxy, ya que leerá el valor del campo Via y verá la IP privada del ordenador y no sabrá hacia dónde encaminarla. Esto queda

reflejado en la figura respuestaNat.

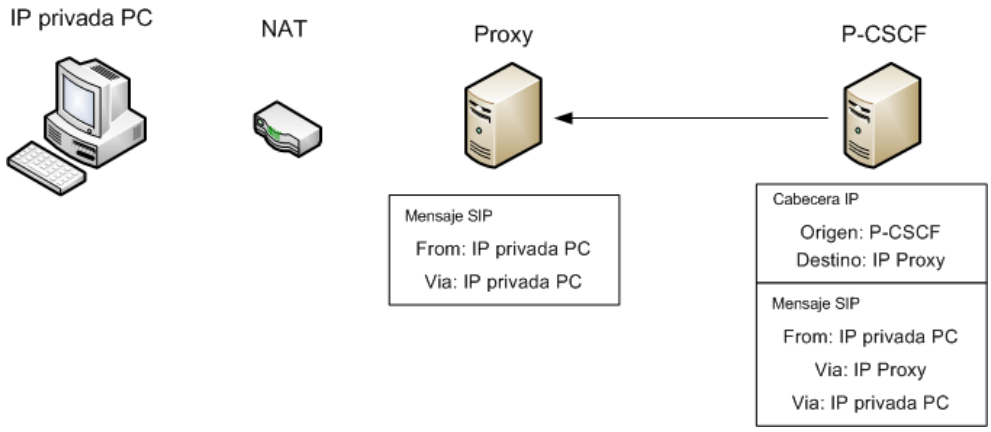

Figura 5.2: Respuesta de un mensaje SIP a un dispositivo colocado tras una NAT.

Este problema tiene dos soluciones:

1. **Cambiar lugar de trabajo**: Esta fue la solución utilizada. El lugar de desarrollo del proyecto se trasladó a un aula de laboratorio de la universidad, donde la conexión a internet me proporcionase una IP publica.

2. **Instalación de una NAT transversal, o NAT-T**: Esta es la mejor solución al problema propuesto y se llevó a cabo en la fase final del proyecto. Una NAT transversal tiene la capacidad de acceder a contenido del mensaje SIP y cambiar la IP privada del ordenador del campo Via por su IP publica. De esta manera, cuando la respuesta le llegue al Proxy, este verá en el campo Via una IP publica que sabrá encaminar.

5.2.2. IPv6

Una vez que se desarrolló la funcionalidad del programa en el proyecto *Functionality*, sección 3.2, llegó el momento de desarrollar la aplicación para un ordenador y para un dispositivo móvil.
A la hora de probar el funcionamiento de la aplicación desde el dispositivo móvil, era necesario que el dispositivo estuviese conectado a Internet a través de una red WiFi. Cuando se probó la aplicación desde las redes públicas accesibles desde la Universidad, se comprobaba que la red no contestaba a los

mensajes. Además la ejecución del programa desde algunos ordenadores desde dentro de la misma universidad tenían el mismo problema.
El problema que producía este comportamiento era que el dispositivo que contenía el programa cliente disponía de IPv6. En un principio, este hecho no supone ningún problema, pero la red IMS instalada en la Universidad dispone de un filtro para este tipo de direcciones. Por ello, cuando llega una solicitud con una dirección IPv6 la red descarta el mensaje ya que no dispone de mecanismos para el tratamiento de estas IPs.
La manera en la que se solventó este problema fue incluyendo mecanismos, en el programa cliente, que comprobasen la versión de la IP asociada al dispositivo. En caso de que el dispositivo esté utilizando una IPv6, el programa debe de avisar del error y no podrá ser lanzado.
A pesar que esta solución funciona, no es la más acertada ya que limita nuestra aplicación y la hace más dependiente de la red destino. Nuestra aplicación debería ser general y por tanto, debería utilizarse para cualquier tipo de red. La mejor solución a este problema sería implementar los mecanismos necesarios en la red IMS para el soporte de IPv6. Esta solución no se ha implementado ya que se escapa de los objetivos de este proyecto.

5.2.3. Autenticación en el servidor

Cuando se inicia la sesión en el programa cliente, lo primero que hace el programa es autenticar al usuario en la red de destino. Para ello, el cliente envía un mensaje INVITE a la red. Si el perfil del usuario establece el uso de una contraseña para autenticarse en la red, la respuesta que obtendrá el usuario será un *401- Unauthorized*. En dicha respuesta se añade un reto que utiliza el usuario junto con su contraseña, y algunos otros datos, para calcular la respuesta y autenticarse en la red. Para más información sobre el método de autenticación se recomienda la lectura de la sección 3.2.3.

Pero en el método de autenticación se produjo un problema que mantuvo el desarrollo del proyecto paralizado durante varios días. El motivo era que el usuario no se autenticaba, a pesar de que la contraseña y el nombre de usuario estaban escritos correctamente. Una primera solución fue comprobar que el cliente realizaba correctamente el cálculo de la respuesta al reto enviado por la red. Para ello, se investigó el método de autenticación *HTTP Digest* y se analizaron los mecanismos que tiene la red de destino para autenticar a un usuario una vez recibido la respuesta al reto planteado. Esto último se pudo hacer porque la red IMS instalada es de código abierto y podemos acceder al código de la aplicación.

A pesar de que el programa cliente tenía bien implementado el mecanismo de cálculo de la respuesta, el problema radicaba en que dicho cálculo utilizaba una codificación en ASCII mientras que la codificación necesaria debe ser UTF-8. Por lo tanto la solución era trivial y consistía en codificar determinadas cadenas de caracteres a la nueva codificación.

5.2.4. Reproductor multimedia

Como ya se ha explicado en el capítulo 4, el mayor problema que ha tenido la realización de este proyecto es el no disponer de un reproductor multimedia para dispositivos móviles que se adecúe a las necesidades de este proyecto. Es un problema bastante importante ya que el objetivo primordial del proyecto era el desarrollo de un cliente de IPTV para dispositivos móviles, y por tanto, el no disponer de reproductor para dicho dispositivo provocaba que no se pudiera validar el correcto funcionamiento de la aplicación.
A pesar de que se ha intentado compilar el reproductor VLC para Windows CE, la mejor solución para probar la aplicación consistió en el desarrollo de la herramienta para ordenadores personales. Ya que en dicha plataforma sí se disponen de reproductores que se adecúen a nuestras necesidades.

Capítulo 6

Pruebas

Una vez descrito tanto el desarrollo como la historia del proyecto, en este capítulo se describirá el plan de pruebas que se ha llevado a cabo en este proyecto. El plan de pruebas es muy importante en cualquier proyecto, ya que te va a determinar la robustez y fiabilidad de un programa; y ambos factores influirán en la aceptación del programa por parte de los usuarios.

Para afrontar el plan de pruebas de este proyecto, se han analizando los diferentes factores que pueden repercutir en la ejecución del programa. Una vez que se conozcan dichos factores, se deben implementar los mecanismos que limiten su repercusión.

Una vez analizado el proyecto, los factores que pueden influir en la ejecución del programa son los siguientes:

- **Parámetros de entrada**: El programa debe asegurarse que los parámetros de entrada necesarios para la ejecución están introducidos, y además, contienen valores correctos.
- **Conexión a Internet**: Otro factor importante es la conexión a Internet, el programa debe de comprobar que el dispositivo está conectado a Internet y que algunos parámetros de dicha conexión son los correctos.

A continuación se explicarán las medidas tomadas en relación a los factores que se acaban de describir.

6.1. Parámetros de entrada

Antes de comenzar la ejecución del programa, este debe de asegurarse que dispone de los parámetros requeridos y que además dichos parámetros

contienen valores, en teoría, aceptables. Se ha de hacer hincapié en el significado de áceptable', ya que en ningún momento el programa puede determinar si un parámetro contiene el valor correcto o no; simplemente puede determinar que el formato de dicho parámetro es correcto o que, efectivamente, hay un valor introducido.

Existen parámetros como el nombre del reproductor que se utilizará, la carpeta donde se almacenará el fichero de detalles, o el canal del servicio, que son presentados en unas listas que son rellenadas por el propio programa. Este hecho nos da la seguridad de que el valor de dicho parámetro va a ser correcto; sin embargo, el sistema debe comprobar que se ha seleccionado un valor de estas listas. Para determinar esto, el sistema intenta acceder al valor seleccionado por el usuario y en el caso de que salte una excepción significará que no hay ningún valor seleccionado.

Otros parámetros, como el nombre de usuario o la dirección privada, son introducidos por el usuario y tienen el formato de una dirección de email:

nombreDeUsuario@redDeOrigen

En este caso, primero se comprueba que el usuario a introducido algún valor y después se comprueba el formato de dicho valor. Para validar el formato, el programa utiliza una expresión regular. Si el valor introducido coincide con la expresión regular, significará que la dirección de email es correcta. La expresión regular utilizada para validar estos parámetros es la siguiente:

[a-z0-9]+([.][a-z0-9]+)@[a-z0-9]+([.][a-z0-9]+)+*

Y a continuación pasaré a explicar cada uno de sus miembros:

- *[a-z0-9]+[.][a-z0-9]+*: Con esta parte nos aseguramos que el inicio del parámetro es una cadena de caracteres formada por letras y números. La longitud de esta cadena será como mínimo un caracter, como indica el símbolo '+ál final de la secuencia. Además esta cadena puede ir acompañada de un punto y de otra cadena, tantas veces como se quiera.
- *@*: Con este otro miembro nos aseguramos que debe existir, obligatoriamente, una arroba en el parámetro introducido.
- *[a-z0-9]+*: Al igual que en la primera parte de la expresión, con este miembro nos aseguramos que después de la arroba tenemos una cadena alfanumérica que forma parte del dominio del parámetro.

- *([.][a-z0-9]+)+*: Finalmente, con *[.][a-z0-9]+* nos aseguramos la existencia de un punto después de la arroba y seguidamente tendremos otra cadena alfanumérica. Para expresar que esto se puede repetir, pero como mínimo debe aparecer una vez se pone todo lo anterior entre paréntesis y se añade el simbolo '+ál final.

Como se puede observar, el programa no puede validar el contenido de la cadena introducida ya que no existe ninguna restricción referente a dicho contenido.

En lo referente al valor de los campos de Red y Servicio, el programa comprueba que dichos campos contienen algún valor introducido y que dicho valor contiene caracteres alfanuméricos y no sólo espacios en blanco. Además para el contenido del campo Red, es fácil establecer la expresión regular que compruebe el contenido de dicho campo. La expresión utilizada para validar este campo es:

*[a-z0-9]+[.][a-z0-9]+([.][a-z0-9]+)**

Existe un parámetro que no es validado por el programa, el cual se corresponde con la contraseña o *password* del usuario. Los motivos por el que no se comprueba este valor son los siguientes:

1. El establecimiento de una contraseña no es algo obligatorio para todos los usuarios, por lo que pueden existir usuarios que no tengan contraseña y, por tanto, no tengan que introducir ningún valor.

2. No existen reglas que determinen el formato de la contraseña. Por lo que, de momento, una contraseña puede ser cualquier combinación de caracteres alfanuméricos.

Los errores relativos a la contraseña serán descubiertos cuando el programa intente establecer la comunicación con el servidor. Primeramente, el programa intentará registrar al usuario y si dicho usuario tiene asociado una contraseña, el servidor la pedirá con una respuesta *401 - Unauthorized*. Una vez recibida dicha respuesta, el programa calculará una respuesta, o *digest*, utilizando la contraseña introducida por el usuario y un texto, *realm* que envía el servidor. Esta nueva respuesta será enviada al servidor y si este contesta con otro mensaje *401 - Unauthorized*, el programa determinará que hay un error con la contraseña introducida por el usuario. Dicho error puede deberse a que el usuario la ha escrito mal o que no la ha escrito.

Por último, los parámetros de configuración del fichero de detalles o *log*, sólo se validarán cuando la casilla correspondiente a la creación de dicho fichero esté activa. En este caso, las carpetas donde se almacenará dicho fichero se muestran en una lista, por lo que sólo hay que comprobar que el usuario ha seleccionado una carpeta. En cambio, la validación del nombre del fichero es algo más compleja.

Para comenzar, el programa debe determinar si el usuario ha escrito algo o no, en el caso de que no haya nada escrito el programa asignará un nombre por defecto para este fichero el cual es *IPTVClient-Log.txt*. En el caso de que el usuario haya escrito un nombre, el programa debe comprobar si ha escrito alguna extensión para dicho fichero. Para ello, comprueba la existencia del punto, '.', que indica el comienzo de la extensión. Si el nombre introducido no tiene extensión, el programa le asignará la extensión *.txt* correspondiente con archivos de texto. En cambio, puede ocurrir que el usuario introduzca un nombre con una extensión. En esta situación, el programa debe asegurarse de que la extensión introducida por el usuario se corresponde con el contenido del fichero; ya que surgirían problemas si se almacena un fichero de texto con la extensión de un fichero de audio o un clip de video, por ejemplo. Para evitar este problema, el programa elimina la extensión introducida por el usuario y se queda con el nombre del fichero al cual le concatena la extensión de archivo de texto.

6.2. Conexión a Internet

Este factor es clave para la correcta ejecución del programa; ya que aunque todos los parámetros anteriormente descritos son correctos, el programa no funcionará si el dispositivo no está conectado a Internet y las características de esta conexión no satisfacen las necesidades de la red contra la que ejecutaremos el programa.
Cuando el usuario introduzca todos los datos y desee iniciar la sesión, el programa debe comprobar que el dispositivo está conectado a Internet. Para ello, el programa realiza una petición http a la página *www.google.com*. Si el código del estado de la respuesta es OK, significa que el dispositivo está conectado a Internet. En cambio si el estado no es OK o se produce un error al acceder a dicho estado, significa que el dispositivo no está conectado.
Una vez validada la conexión a Internet, se deben de comprobar que las características de dicha conexión satisfacen las características de la red contra la que vamos a ejecutar el proyecto. Como ya se describió en la sección 5.2, el sistema debe comprobar si el dispositivo está conectado a

Internet usando IPv4. Para ello, el programa intenta almacenar la IPv4 asociada a la conexión, en caso de no encontrarla se produciría un error.
Además, durante el desarrollo de la práctica hubo un tiempo en el que no se disponía de una NAT transversal, por lo que el programa debía verificar que la IPv4 asignada al dispositivo se correspondía con una IP pública y no privada. Esta validación es trivial ya que las direcciones privadas están definidas en unos rangos concretos de direcciones IP; por lo que el programa solo tenía que comprobar que la dirección que tenía el dispositivo no se encontraba en dichos rangos. A pesar de la instalación, a posteriori, de la NAT transversal, la funcionalidad encargada de validar la IP no ha sido eliminada de la aplicación. Esta funcionalidad, no es ejecutada en ningún momento, pero no ha sido eliminada ante la posibilidad de que en un futuro pueda ser necesaria.

6.3. Comportamiento ante errores

El comportamiento de la aplicación ante cualquiera de los errores descritos anteriormente es siembre el mismo. Primero se muestra por pantalla un mensaje donde se especifica el error, además si el usuario ha seleccionado la opción de utilizar el fichero de detalles, se crea una entrada en dicho fichero describiendo el error producido. Una vez hecho esto la ejecución del programa se para y se devuelve el control del programa al usuario, que tiene la posibilidad de solventar el error y continuar la ejecución del programa o cerrar el programa.

6.4. Descripción de la batería de pruebas

En esta sección se describirá la batería de pruebas que se ha utilizado para comprobar la funcionalidad de la aplicación y los resultados obtenidos ante dicha batería. Como ya se ha explicado en la sección anterior, los dos grandes factores que pueden influir en la ejecución de la aplicación son el formato de los parámetros de entrada y la conexión a internet del dispositivo.

6.4.1. Prueba de la conexión a Internet

Para probar la conexión a Internet no se ha programado una batería de pruebas ya que considero que se tarda menos tiempo lanzando la aplicación varias veces, unas veces teniendo el dispositivo conectado y tras no, que programando una batería de pruebas que se encargen de lanzar la aplicación

bajo diferentes estados de la conexión. Por tanto, se ha lanzado de forma manual la aplicación, en su versión para ordenador, diez veces. La mitad de estas ejecuciones se realizaron con el dispositivo conectado a internet y la otra mitad, con el dispositivo conectado.

El objetivo de esta prueba es ver que el programa es capaz de detectar si el dispositivo está conectado a internet. En todos los casos en los que el dispositivo se encontraba conectado a Internet, la aplicación ha iniciado el establecimiento de la sesión con el servidor. Además, cuando el dispositivo no ha estado conectado se mostraba por pantalla y se escribía en el fichero de detalles el mensaje que anuncia al usuario que el dispositivo no estaba conectado. Seguidamente, el control del programa volvía al usuario el cual podía volver a lanzar la aplicación o cerrarla.

Después de estos resultados se testeó la versión para dispositivos móviles. Esta vez, sólo se hicieron dos pruebas, una conectado y otra desconectado, aunque no hacía falta esta prueba ya que el procemiento encargado de comprobar la conexión es común a ambas versiones. El resultado que se obtuvo fue el mismo que se ha descrito en el párrafo anterior.

6.4.2. Pruebas contra los URI SIP

Un primer parámetro que se debe validar es el correcto formato de el uri privado y público del usuario, ambos valores tienen el mismo formato, el cual se valida con el método *validateAddress* de la clase *GUI.cs*. Este método se ha declarado como estático para poder llamarlo sin necesidad de crear una instancia del objeto GUI. Con esto, lo que hemos hecho ha sido modificar el código de la clase *MainFile.cs*, para que antes de lanzar la aplicación se lance una batería de pruebas contra los URI SIP.
Para crear la batería de pruebas se crean dos arrays, uno de Strings y otro de booleanos, con la misma dimensión. La idea es que en el array de Strings se almacenen todos los valores que vamos a comprobar y en el array de booleanos se almacene el resultado esperado de la validación. Esto quiere decir que si en la posición cero del array de Strings aparece la cadena "prueba", en la posición cero del array de booleanos aparecerá el resultado que debería dar la validación de la cadena "prueba".
Después, mediante un bucle recorreremos ambos arrays ejecutando el método de validación sobre los Strings y comparando el resultado obtenido con el resultado esperado. Si ambos resultados coinciden es que el test es correcto, en caso de que ambos resultados difieran significará que se ha encontrado una cadena que no cumple la regla definida.

Teniendo en cuenta la expresión regular diseñada para validar los URI SIP se ha diseñado la siguiente batería de pruebas:

Cadena de Prueba	**Resultado Esperado**
alice@carissimi.gast	True
alice.alice@carissimi.gast	True
alice.alice@carissimi.gast.it	True
alice@carissimi	False
@carissimi.gast	False
alice..@carissimi.gast	False
a_@carissimi.gast	False
alice@carissimi..gast	False
alice@carissimi..	False

Hay que aclarar que lo que estamos validando son los métodos encargados de comprobar el correcto formato de los parámetros. Los primeros tres resultados son correctos pero sólo en formato, si lanzamos la aplicación con cualquiera de estos tres datos obtendremos la misma respuesta del servidor, la cual dice que el usuario es desconocido. Esto se debe a que esos valores no están registrados en el HSS de la red contra la que lanzamos la aplicación.

Una vez, lanzada la batería de pruebas comprobamos que de todos los valores se obtienen los resultados esperados, como muestra la figura 6.1.

```
Test de alice@carissimi.gast correcto
Test de alice.alice@carissimi.gast correcto
Test de alice.alice@carissimi.gast.it correcto
Test de alice@carissimi correcto
Test de @carissimi.gast correcto
Test de alice..@carissimi.gast correcto
Test de a_@carissimi.gast correcto
Test de alice@carissimi..gast correcto
Test de alice@carissimi.. correcto
```

Figura 6.1: Resultado de la batería de pruebas contra URI SIP.

6.4.3. Pruebas contra los dominios

En esta sección se describe la batería de pruebas lanzada contra el nombre de la red que contiene el servicio que solicita el usuario. El método de actuación es exactamente igual al descrito en la sección anterior, sólo que

ahora el método que controla el correcto formato de los dominios se llama *validateDomain*.

La batería de pruebas que se ha lanzado con su correspondiente salida espererada, se muestra en la siguiente tabla:

Cadena de Prueba	**Resultado Esperado**
carissimi.gast	True
carissimi.gast.it	True
carissimi.gast.it.uc3m	True
carissimi.gast.it.uc3m.es	True
carissimi.gast.	False
car_ssimi.gast	False
carissimi.g_st	False
carissimigast	False
.gast.it.uc3m.es	False

La figura 6.2 muestra los resultados de la batería de pruebas y como podemos observar la función encargada de la validación del nombre de la Red funciona correctamente.

```
Test de carissimi.gast correcto
Test de carissimi.gast.it correcto
Test de carissimi.gast.it.uc3m correcto
Test de carissimi.gast.it.uc3m.es correcto
Test de carissimi.gast. correcto
Test de car_ssimi.gast correcto
Test de carissimi.g_st correcto
Test de carissimigast correcto
Test de .gast.it.uc3m.es correcto
```

Figura 6.2: Resultado de la batería de pruebas contra el nombre de la red.

6.4.4. Pruebas contra el nombre del servicio

El siguiente parámetro que se validará será el nombre del servicio que solicita el usuario. La validación de dicho parámetro corre a cargo del método *validateNormalString* y es la validación más sencilla ya que no existe ningún formato específico para este valor. Por lo tanto la única restricción es que el usuario debe escribir una cadena alfanumérica de longitud mayor o igual a un caracter.

Dados los escasos test que se pueden realizar contra estre parámetro, no haría falta programar una batería de pruebas; en cambio, para mantener la

misma línea de trabajo validaremos este valor de la misma manera que en las secciones anteriores. Estos son los valores de las pruebas.

Cadena de Prueba	Resultado Esperado
iptv	True
i	True
89	True
servicio45	True
i_tv	False
servicio4.6	False

De nuevo, el funcionamiento del método encargado de validar el formato es correcto como demuestra la figura 6.3

```
Test de iptv correcto
Test de i correcto
Test de 89 correcto
Test de servicio45 correcto
Test de i_tv correcto
Test de servicio4.6 correcto
```

Figura 6.3: Resultado de la batería de pruebas contra el nombre del servicio.

6.5. Tiempos de Conexión con el servidor

En esta sección se van a presentar los tiempos que ha tardado la aplicacion en conectarse con el servidor. Sólo se ha podido probar la versión para ordenador, debido a que un problema en el servidor imposibilitó que la aplicación se conectara al servidor cuando debería comenzar la toma de datos de la versión para dispositivos móviles.

En cuanto al tipo de conexión, se han probado tres tipos de conexiones:

- Conexión Wifi desde dentro de la Universidad. La red a la cual se conectó el dispositivo es *gast.wifi*.

- Conexión de Ethernet desde dentro de la Universidad. En este caso se lanzo la aplicación desde un ordenador de los laboratorios.

- Conexión de Ethernet desde fuera de la Universidad. Finalmente, se tomaron los tiempos que tardaba la aplicación en conectarse, si el dispositivo estaba conectado por Ethernet a una red externa al dominio de la Universidad.

Para el cálculo de los tiempos se ha lanzado la aplicación y se han tenido el cuenta los milisegundos iniciales y finales de la conversación con el servidor. Este experimento se ha repetido veinte veces para tener una toma lo suficientemente extensa como para poder sacar conclusiones.

Los resultados obtenidos en el caso de la conexión Wifi desde dentro de la universidad son los siguientes:

Milisegundo Inicial	**Milisegundo Final**	**Diferencia**
609	1898	2340
355	588	1233
15	232	1217
260	508	1248
461	741	1280
760	976	1216
760	108	1294
362	610	1248
937	962	1544
105	369	1264
988	566	1295
366	646	1280
23	645	1622
869	328	1295
278	300	1872
573	852	1279
925	410	1280
803	6090	7067
52	424	1372
846	188	1248

Como podemos observar en esta tabla el tiempo medio que tarda el ordenador en conectar con el servidor utilizando la red wifi del laboratorio es de 1674 milisegundo, que es algo más de un segundo y medio.

Seguidamente se presentarán los datos relativos a la ejecución de la aplicación desde un ordenador conectado por Ethernet desde dentro de la Universidad:

Milisegundo Inicial	Milisegundo Final	Diferencia
515	2453	1938
578	1953	1375
812	2234	1422
93	1281	1188
156	1390	1234
31	1218	1187
593	1843	1250
46	1250	1204
46	1296	1250
437	1640	1203
390	1609	1219
484	1703	1219
531	1750	1219
31	1250	1219
531	1781	1250
640	1843	1203
968	2250	1282
593	1812	1219
281	1500	1219
671	1921	1250

En este caso, el tiempo medio de conexión es de 1277 milisegundos. Este valor tan bajo dependerá de la conexión de la que disponen los ordenadores de los laboratorios y su velocidad de navegación. A pesar de ello el valor demuestra que el usuario no tendrá que esperar en exceso y podrá disfrutar del servicio solicitado sin apenas esperar al establecimiento.

Por último, en la siguiente tabla se muestran los tiempos que tarda la aplicación conectada por Ethernet a una red externa de la Universidad.

Milisegundo Inicial	Milisegundo Final	Diferencia
687	2174	1487
718	2176	1458
347	1805	1458
442	1914	1472
169	1598	1429
391	1966	1575
479	1944	1465
685	2189	1504
252	1710	1458
506	1946	1440
721	2211	1490
24	1486	1462
526	1981	1455
773	2220	1447
501	1973	1472
636	2219	1583
178	1640	1462
105	1577	1472
928	2400	1472
583	2095	1512

Los valores obtenidos en esta última ejecución muestran unos tiempos más lentos que la ejecución anterior pero aún así son unos valores bastante aceptables. El valor medio de esta última ejecución es de 1478 milisegundos y se corresponde con el valor intermedio de las tres mediciones realizadas.
Hay que tener en cuenta que este tiempo es el que se puede obtener desde cualquier ubicación externa a la universidad y que la conexión con la que se ha probado no es una de las conexiones más rápidas que existen hoy en día.
Entre los datos obtenidos en este apartado hay que destacar que la conexión más lenta ha sido la de la red Wifi, este dato no debe de sorprendernos ya que la conexión Wifi es algo más lenta que la conexión através del cable de Ethernet. Además la conexión Wifi depende de obstáculos que pueden interferir en la señal, como pueden ser las paredes.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se analizarán los resultados obtenidos al finalizar el desarrollo del proyecto. Se analizarán los aspectos positivos del proyecto así como los negativos, para identificar cuáles pueden ser las futuras vías de desarrollo de este proyecto.

7.1. Aspectos positivos

En cuanto a los aspectos positivos del proyecto hay que destacar el desarrollo del cliente de IPTV. El programa desarrollado es capaz de registrar al usuario en la red y de establecer la sesión con el servidor de IPTV. Además el sistema es robusto, ya que se han analizado los factores que pueden influir en su ejecución y se han programado, y comprobado, los procedimientos necesarios para que si se produce algún error, el usuario tenga la opción de solventar el error y proseguir con la ejecución o cerrar la aplicación. Para más información sobre el tratamiento de errores se recomienda la lectura del capítulo 6

Por otro lado, el diseño por módulos del código del proyecto *Functionality*, descrito en la sección 3.2, facilita la adhesión de nuevos módulos a la funcionalidad del proyecto. Lo que provoca que el código sea ampliable con nuevos módulos y nuevas funcionalidades. En el caso de añadir nuevos módulos sólo habría que cambiar el código de las clases que sirven de unión entre los proyecto, las cuales son *Core* del proyecto específico del dispositivo y *MessageHandler* en el caso de que se añadan nuevos protocolos de comunicación.

Además, este diseño en módulos permite la reutilización de los módulos que desarrollados en esta aplicación a nuevos proyectos. Por ejemplo, si se quisiera hacer un proyecto que necesite comunicarse con un servidor, se podría reutilizar el módulo de conectividad, representado por el fichero *ConnectionController.cs*, en el nuevo proyecto sin necesidad de cambiar ninguna línea de código de dicho módulo.

Hay un aspecto positivo muy importante, que no se había previsto al iniciar este proyecto y que lo ha proporcionado el hecho de aislar la funcionalidad del cliente IPTV en una librería llamada *Functionality.dll*. El hecho de tener la funcionalidad aislada nos permite la utilización del cliente en casi cualquier tipo de dispositivo. Para ello, necesitaremos crear un proyecto específico para el dispositivo objetivo que utilice la librería mencionada anteriormente. Una vez hecho esto, el nuevo proyecto deberá contener una interfaz similar a la utilizada en los proyectos *IPTVClient* e *IPTVClient-PC* y una clase *Core* con el mismo código que poseen estos dos proyectos. Los único habría que hacer sería el ajustar los reproductores y las rutas de estos reproductores a la lista de reproductores disponibles en el nuevo dispositivo. Esta es una ventaja proporcionada por la utilización del patrón arquitectónico *Modelo Vista Controlador* descrito en la sección 3.4. Con el uso de este patrón, mantenemos una independencia entre la interfaz, la lógica y los datos del programa. Lo que permite la modificación de cualquiera de estos sin que ello influya, excesivamente, en los otros dos.

Por último mencionar que el diseño claro y sencillo de la interfaz que se ha utilizado en estos dos proyectos, hace que la utilización de los programas sea algo cómodo, fácil de usar y fácil de aprender. Además la etiqueta de Opciones que permite personalizar las ejecuciones del programa le añade un valor añadido al programa que será bien recibido por parte del usuario.

7.2. Aspectos negativos y posibles líneas de desarrollo

El aspecto negativo que se le encuentra al código es que éste es muy dependiente de la red que posee el servicio. Cuando el usuario inicia la sesión, los mensajes deben ir dirigidos al *P-CSCF* por lo el programa concatena el nombre de la función al nombre de la red introducida y se envía. Lo que pretendo explicar es que el código no posee ningún mecanismo para conocer la arquitectura de la red y por tanto, si en un futuro la

red cambiase el programa debería seguir funcionando igual ya que este debería conocer, cuando se ejecute, a qué nodo de la red debe enviar los mensajes. Si en un futuro, el nodo que contiene la función de *P-CSCF* en lugar de llamarse *pcscf.carissimi.gast.it.uc3m.es* se llamase, por ejemplo, *inicio.carissimi.gast.it.uc3m.es* el código en lugar de descubrir que debe enviarlo a *inicio* lo enviará a *pcscf* por lo que programa no funcionará.

Además el proyecto está diseñado para proporcionar un servicio de IPTV y la lista de canales se completa si y solo si, el nombre de la red es *carissimi.gast.it.uc3m.es* y el servicio es *iptv*. Sin embargo, esta lista de canales se completa porque está establecido así en el código. Es decir, a lo largo del código hay un momento que dice que si el servicio es *iptv* y la red es *carissimi.gast.it.uc3m.es*, entonces la lista de canales se completa con *channel1*, *channel2* y *channel3*, independientemente de si son esos los nombres de los canales o el número de canales. Sería muy interesante, llegados a este punto el desarrollo de un sistema que permitiese descubrir los servicios de una determinada red, para así tener un código más independiente.

Un problema que se ha tenido durante el desarrollo de este proyecto es que la implementación del servidor no es fiel, en la totalidad, al estándar. Esto provoca problemas de compatibilidad cuando se intenta crear un cliente basado en dicho estándar. Este problema se puso de manifiesto durante el desarrollo del cliente SIP en el cual se produjeron algunos problemas con las cabeceras SIP, debido a que el servidor necesitaba cabeceras que, en un principio, no debían ser necesarias. De esto se deduce la falta de fidelidad del software del servidor al estándar que intenta implementar que provoca un ligero ajuste en el programa diseñado.

En lo que respecta al tema del reproductor, queda mucho trabajo por hacer debido a que no ha sido posible migrar la aplicación a la plataforma que nos interesa, Windows CE. Esto ha impedido la utilización del sistema de PPV para acceder a determinados contenidos. El principal problema que se ha encontrado a lo largo del proceso de migración de la aplicación es la falta de documentación y la poca calidad en la documentación existente. Con esto me refiero a que acompañando al código tenemos un fichero de instalación del programa para diferentes aplicaciones, y en el caso de Windows CE, este fichero describe el proceso de instalación utilizando unas herramientas que se encuentran en desuso.

Además, aparecen otros problemas que son los errores que poseen las últimas versiones de código a pesar de que estas versiones son publicadas

como versiones estables, ya que las últimas versiones son 1.0.0 y 1.0.1. Esto obliga a utilizar versiones más antiguas o a utilizar las nuevas versiones de código con la tarea presente de revisión y corrección de estos errores.

Tras todo lo expuesto en esta sección, si se debe recomendar una futura línea de desarrollo dicha línea es el proceso de migración del reproductor a la plataforma Windows CE, y el desarrollo de una interfaz que permita la fácil reproducción de los servicios que solicita el usuario. Una vez que se disponga del programa portado a la plataforma con el sistema de pago por visión, podría ser muy interesante la integración del propio reproductor dentro de la interfaz diseñada en este proyecto. Con esto, mejoramos la usabilidad de la aplicación y evitamos que se abra una nueva aplicación para reproducir el servicio. Si tuviésemos todo junto en la misma interfaz, la ejecución de la aplicación sería más clara y más sencilla.

En lo referente al sistema de pago por visión, no estoy en posición de dar una valoración ya que no he tenido el placer de trabajar en dicho área debido a la imposibilidad de portar el reproductor. Pero tengo conocimiento de que dicho sistema ha sido integrado con el programa VLC en anteriores proyectos desarrollados en la universidad Carlos III, obteniendo resultados muy satisfactorios.

Apéndice A

Manual de instalación

El objetivo de este apéndice es proporcionar una guía sobre cómo instalar el programa *IPTV Client*. Durante el estudio de este proyecto se han desarrollado la aplicación para dos plataformas diferentes; por ello, en este capítulo se describirá la instalación en ambas plataformas. Hay que recordar que el objetivo principal del proyecto es el desarrollo de la aplicación para dispositivos móviles, es por ello, que el proceso de instalación en dispositivos móviles sea más extenso y detallado que el proceso de instalación en un ordenador con Windows.

A.1. Instalación en un dispositivo móvil con Windows Mobile

En esta sección se detallará el proceso de instalación de la aplicación en un dispositivo móvil con Windows Mobile como sistema operativo.
Lo primero que se debe hacer para iniciar el proceso de instalación, es almacenar el instalable dentro del dispositivo. Esto se puede hacer conectando el dispositivo al ordenador y guardándolo en la memoria del teléfono, o utilizando una tarjeta de memoria. Para ilustrar esta guía utilizaremos una tarjeta de memoria.
Una vez que tengamos los ficheros en el dispositivo y dicho dispositivo esté encendido, deberemos entrar en el Explorador de Archivos o *File Exporer* y localizar la carpeta que acabamos de almacenar. Como podemos ver en la figura A.1, la carpeta contiene tres ficheros. De estos tres ficheros el que nos interesa es *IPTV-Client* que tiene un tamaño de 51.9 KB.

Una vez que comienza el asistente de instalación, se le pide al usuario que especifique dónde desea instalar el programa. Como muestra la figura

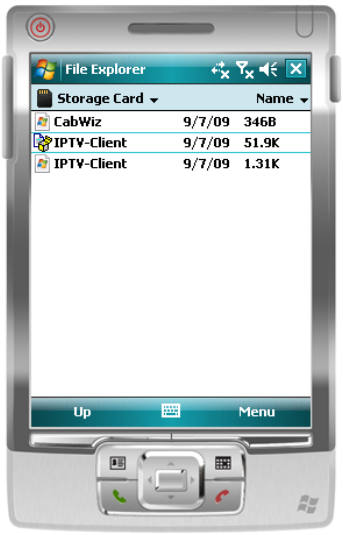

Figura A.1: Contenido de la carpeta de instalación.

A.2, el usuario tiene dos opciones: puede instalar la aplicación en el propio dispositivo o en la tarjeta de memoria. Cabe destacar que este paso sólo se realizará si el dispositivo dispone de una tarjeta de memoria. En el caso de que el dispositivo no disponga de dicha tarjeta, el instalador procederá a instalar la aplicación en el dispositivo.

Cuando el dispositivo finaliza la instalación correctamente, se muestra por pantalla un mensaje que indica que la instalación se ha producido satisfactoriamente, como se puede observar en la figura A.3. Si se produce algún error durante la instalación, aparecerá una ventana emergente en la cual se proporciona una descripción del problema, y en la ventana del instalador se podrá leer *Error al instalar IPTV-Client.CAB*.

Si el usuario desea ejecutar el programa instalado, debe abrir el Explorador de Archivos o *File Exporer* y dirigirse a la carpeta Archivos de Programa o *Program Files*. Si el programa ha sido instalado, en dicho directorio debe aparecer una carpeta llamada *IPTV-Client*, como muestra la figura A.4. Dentro de esta carpeta se encuentra el fichero ejecutable llamado *IPTVClient*. Al ejecutar este fichero se ejecuta la aplicación. La figura A.5 muestra el contenido de la carpeta *IPTV-Client*.

Si durante la instalación de la aplicación el usuario seleccionó instalar la aplicación en la tarjeta de memoria, para ejecutar el programa deberá di-

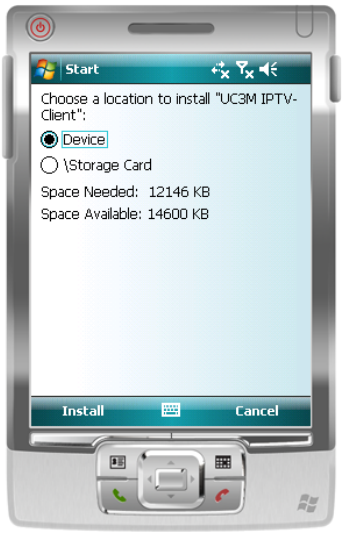

Figura A.2: Selección del destino de la instalación.

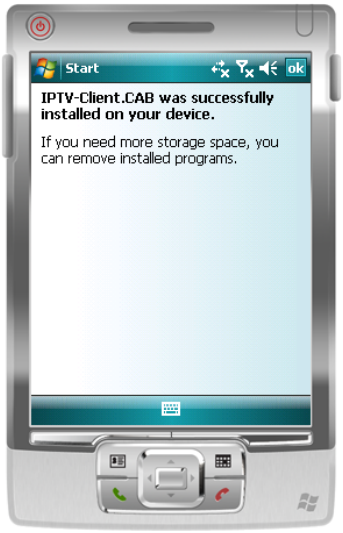

Figura A.3: Mensaje de éxito en la instalación de la aplicación.

rigirse con el Explorador de Archivos a la tarjeta de memoria. En dicho directorio aparecerá una carpeta llamada Archivos de programa, la cual contendrá la carpeta *IPTV-Client* mencionada en el párrafo anterior.

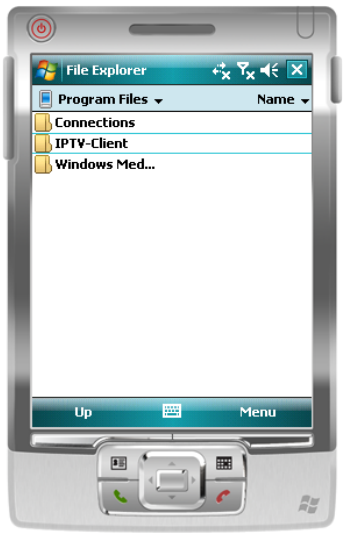

Figura A.4: Contenido del directorio *Program Files* después de la instalación.

Finalmente si se desea desinstalar la aplicación, el usuario deberá entrar en Configuración o *Settings* y seleccionar la pestaña Sistema o *System*. Una vez que el usuario se encuentre en dicha pestaña, deberá seleccionar Quitar programas o *Remove Programs*. En la pantalla aparecerá una lista con todos los programas instalados en el dispositivo. Si la aplicación ha sido instalada correctamente, en dicha lista deberá aparecer el programa *UC3M IPTV-Client* que representa la aplicación. El usuario deberá seleccionar esta entrada de la lista y pinchar sobre el botón Quitar o *Remove*.

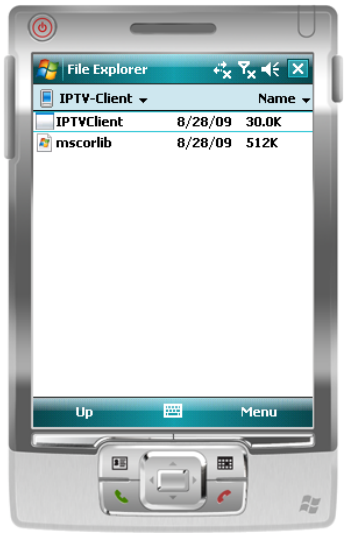

Figura A.5: Contenido de la carpeta *IPTV-Client*.

A.2. Instalación en un ordenador con Windows

El objetivo de esta sección es describir el proceso de instalación de la aplicación *IPTV Client* en un ordenador con un sistema operativo Windows. Como ya ha descrito al inicio de este apéndice, esta sección no será tan detallada como la sección anterior, debido a que el proyecto *IPTVClient-PC* no es objetivo de este estudio; sino que es un proyecto estable de respaldo al objetivo principal.

Para comenzar la instalación de la aplicación, el usuario deberá ejecutar el programa llamado *setup.exe*. Al ejecutar este programa se mostrará un asistente que guiará al usuario a lo largo del proceso de instalación. El paso más importante en la instalación es el que muestra la figura A.6. En dicho paso, el usuario podrá seleccionar la carpeta donde desea instalar la aplicación. Por defecto, el programa se instalará dentro de la carpeta *Program Files* del disco duro. En dicho directorio se creará la carpeta *UC3M* y dentro de esta, *IPTV-Client* contendrá los ficheros necesarios para la ejecución.

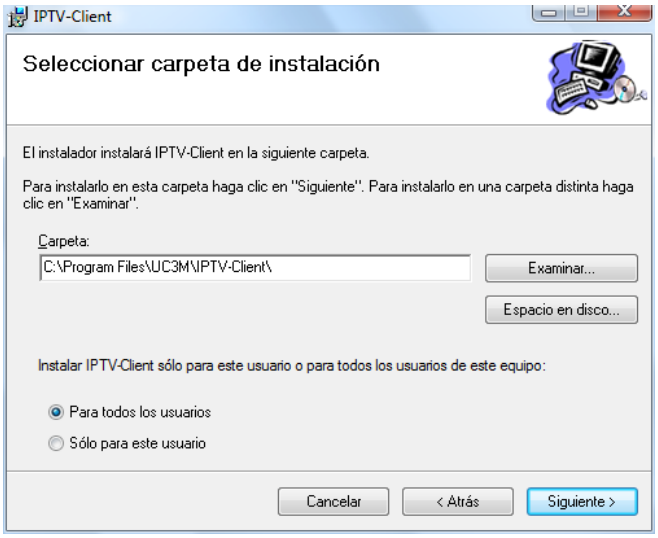

Figura A.6: Selección de la carpeta de destino de la instalación.

Si la instalación se ha realizado correctamente, el asistente mostrará al usuario el mensaje de Instalación completada como muestra la figura A.7.

Una vez que se ha instalado la aplicación existen tres manera de ejecutar la aplicación:

1. Navegando por los directorios hasta llegar a la carpeta donde se ha instalado la aplicación. La carpeta *IPTV-Client* contendrá, entre otros ficheros, el programa *IPTVClient-PC.exe* que será el encargado de ejecutar la aplicación.

2. Cuando se realiza la instalación de la aplicación se creará en el escritorio, un acceso directo a la aplicación llamado *IPTV-Client*. Al hacer click sobre dicho acceso directo se ejecutará la aplicación.

3. Otra manera de ejecutar la aplicación, es haciendo uso del menú Inicio. En dicho menú, en la sección de Programas, debe aparecer una entrada llamada *UC3M*. Dicha entrada contiene un archivo llamado *IPTV-Client* que al seleccionarlo lanzará la aplicación.

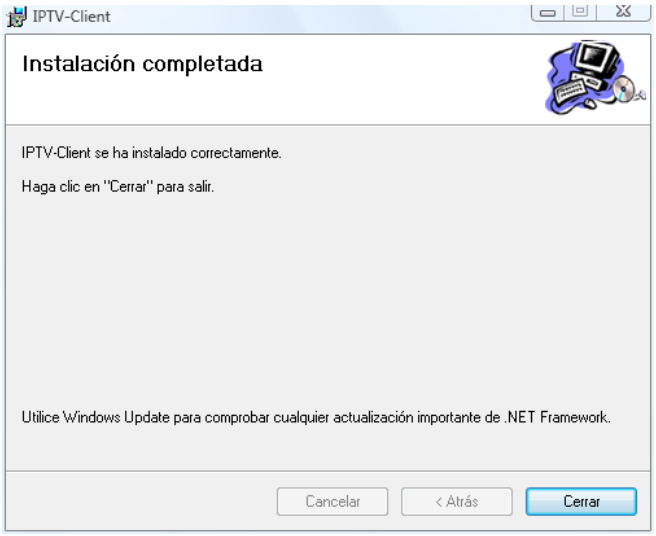

Figura A.7: Instalación satisfactoria de la aplicación para ordenador.

Para terminar, se debe describir el proceso de desinstalación de la aplicación. Si el usuario desea desinstalar la aplicación deberá acceder a la opción Agregar o quitar Programas del Panel de Control[1]. En la lista de programas

[1]Nótese que el nombre de la opción "Agregar o quitar Programas" puede variar dependiendo de la versión de Windows que el usuario esté usando. Por ejemplo, si el usuario utiliza Windows Vista, la opción se llama "Programas y características".

de programas que se muestra, debe aparecer una entrada llamada *IPTV-Client*. El usuario deberá seleccionar esta entrada y pulsar sobre el botón Desinstalar.

Apéndice B

Manual de usuario

En el siguiente apéndice se describirá cómo se debe usar el programa, indicando paso a paso las acciones que debe hacer el usuario y dando una explicación de cada campo de la interfaz.
Como ya se ha descrito anteriormente, este estudio está compuesto por dos proyectos, uno es una versión del programa para ordenador, mientras que el otro es una versión para dispositivos móviles. Como se puede observar en la figura B.1, las interfaces de ambos proyectos son iguales y contienen los mismos campos; por lo tanto, el manual de usuario es el mismo independientemente del proyecto que se esté utilizando.

Para simplificar el manual, se utilizarán únicamente los gráficos correspondientes a la versión para ordenador aunque las descripciones del manual son perfectamente aplicables a la versión para dispositivos móviles.

B.1. Descripción general

La interfaz general del aplicación se compone de cuatro pestañas que ayudan a presentar la información de manera clara y ordenada, y además facilita la utilización de la aplicación. Las cuatro pestañas que tiene la aplicación son las siguientes:

- **Usuario**: Agrupa la información requerida para autenticar al usuario.
- **Red**: Solicita información sobre la red que contiene el servicio que se solicitará.
- **Sesión**: Contiene controles que permiten manejar la sesión.

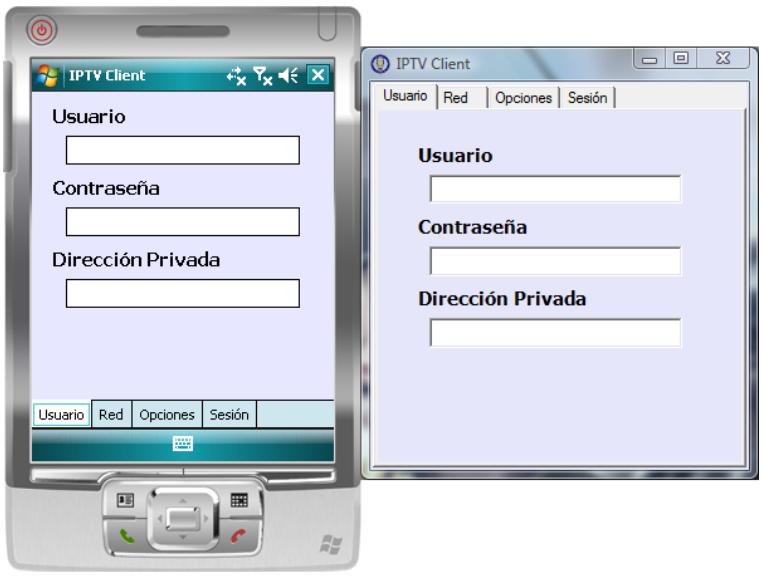

Figura B.1: Comparación entre las interfaces de las aplicaciones para ordenador y para dispositivos móviles.

- **Opciones**: Ofrece características relacionadas con la configuración de la aplicación.

Se proporcionará una descripción más detallada de cada una de estas pestañas en las siguientes secciones.

B.2. Pestaña de Usuario

Esta es la primera pestaña que se muestra al lanzar la aplicación. En ella se solicitan los datos necesarios para poder autenticar al usuario en la red que contiene los servicios requeridos.
Estos datos son los siguientes:

- **Usuario**: Es el nombre del usuario que quiere iniciar la sesión. Este nombre tiene la apariencia de una dirección de mail, por lo que su formato debe de ser:

  nombreDeUsuario@operador

- **Contraseña**: Aquí se debe de introducir la contraseña o password del usuario. Por motivos de seguridad, los caracteres introducidos serán

representados como asteriscos '*'. Este campo solamente será necesario cuando las características de la cuenta del usuario determinen el uso de dicha contraseña.

- **Dirección Privada**: Esta dirección privada tiene el mismo formato que el nombre de usuario introducido anteriormente. Dicha dirección representa la dirección real en la cual el usuario quiere recibir el servicio que va a solicitar.

En la siguiente imagen se puede observar el aspecto de esta pestaña así como los campos anteriormente descritos.

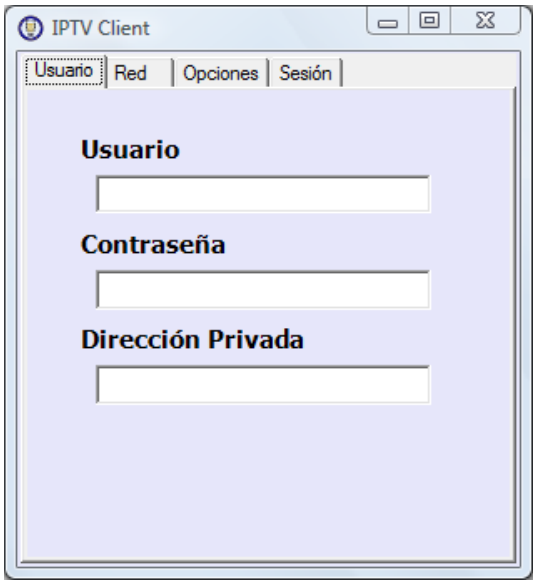

Figura B.2: Descripción de la pestaña de usuario.

B.3. Pestaña de Red

En la segunda pestaña, cuyo nombre es Red, se solicitan al usuario los datos referentes de la red a la que quiere acceder y también se solicitan los datos del servicio que el usuario quiere iniciar.
Como podemos ver en la figura B.3, la información requerida es:

- **Red**: Este campo representa el nombre de la red que contiene el servicio que el usuario solicita.

- **Servicio**: El usuario debe de introducir el servicio que requiere.
- **Canal**: Si el usuario solicita el servicio de televisión *iptv* a la red *carissimi.gast.it.uc3m.es*, esta lista de canales se completará automáticamente. El usuario deberá seleccionar uno de estos canales.

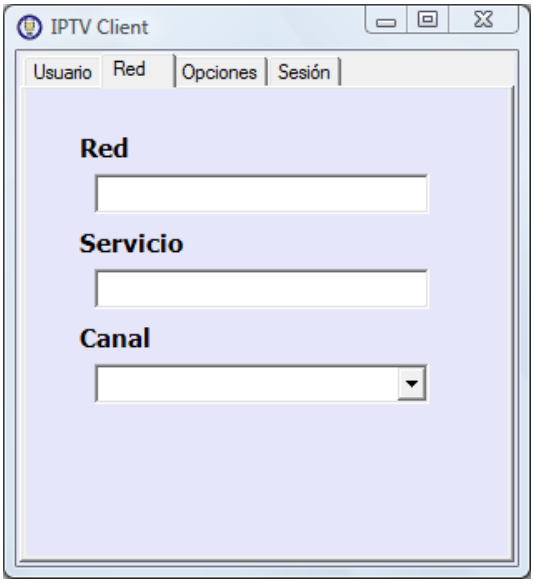

Figura B.3: Descripción de la pestaña de red.

B.4. Pestaña de Opciones

Una vez que se han introducido los valores correspondientes a la identificación del usuario, a la selección de la red y las características del servicio, la aplicación pone a disposición del usuario algunos parámetros que personalizan la ejecución del programa.
El usuario tiene la posibilidad de seleccionar el reproductor con el cual se ejecutará el servicio que ha seleccionado. El tipo de reproductor dependerá de si la ejecución se corresponde con la versión para ordenador o para dispositivos móviles. Las razones de esto se deben a que no existe el mismo abanico de reproductores para ambos dispositivos. El reproductor se debe seleccionar de la lista *Aplicación* y es obligatorio seleccionarlo. Para ver el aspecto de esta pestaña ver la figura B.4.

Además, el usuario tiene a su disposición la posibilidad de crear un fichero de detalles en el cual se almacene una descripción más exhaustiva de los

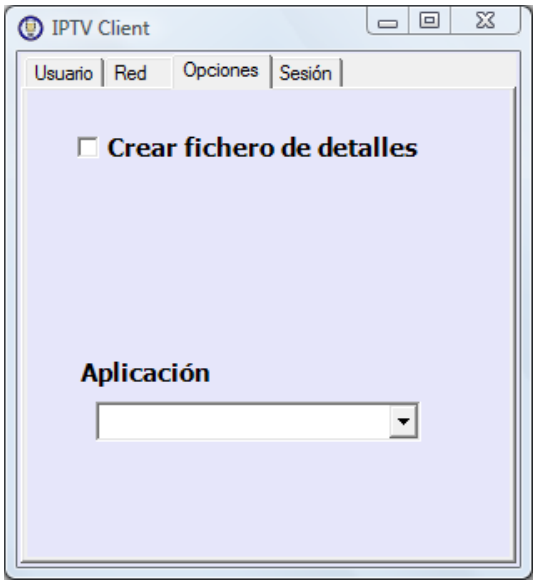

Figura B.4: Descripción de la pestaña de Opciones.

mensajes que la aplicación mostrará por la interfaz. Para poder disponer de dicho archivo, el usuario debe activar la casilla llamada *Crear fichero de detalles*.
En cuanto dicha casilla sea activada, la interfaz cambiará y tendrá el aspecto que presenta la figura B.5.

Como se puede observar, aparecen dos opciones que se son:

- **Nombre del fichero**: El usuario debe de especificar el nombre que tendrá el fichero de detalles. Si dicho nombre tiene extensión, esta será ignorada para evitar incompatibilidades entre el contenido del fichero y su extensión. Si el nombre no tiene extensión, la propia aplicación se encargará de añadirla. Además, si el usuario no especifica ningún nombre, la aplicación creará el fichero con el nombre

  IPTVClient-Log.txt

- **Carpeta del fichero**: Además de especificar el nombre del fichero el usuario deberá seleccionar en qué carpeta quiere que se guarde el fichero. Dicha ubicación queda sujeta a unas carpetas específicas que se pueden seleccionar de esta lista.

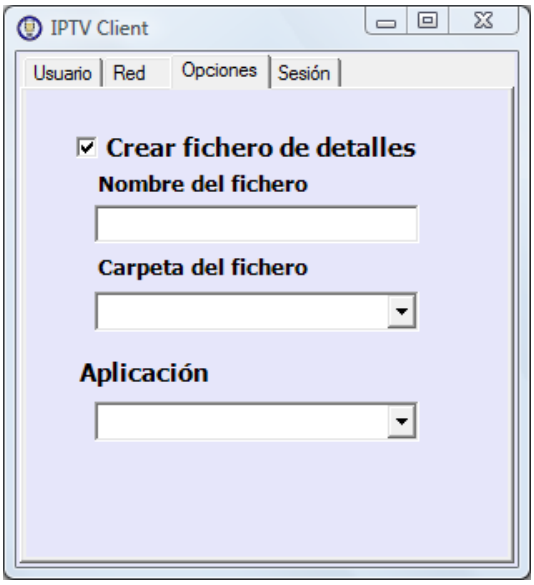

Figura B.5: Detalle de la pestaña de Opciones.

B.5. Pestaña de Sesión

Finalmente, una vez que se han introducidos todos los datos de configuración, el usuario debe ir a la pestaña de sesión para comenzar la reproducción.

En dicha pestaña, ver Figura B.6, aparece el botón *Iniciar Sesión* que inicia la sesión con los parámetros introducidos en las pestañas anteriores. Durante el establecimiento de la sesión, e independientemente de si el usuario desea la creación de un fichero de detalles o no, se mostrarán algunos mensajes en la pantalla a modo de información para el usuario; de la misma manera que si se produjese algún error de configuración también se mostraría en la pantalla. Un ejemplo de esto se puede observar en la figura B.7 en la cual el usuario no ha introducido su nombre de usuario en la pestaña de Usuario, por lo que el programa detecta que el nombre es incorrecto y además indica al usuario que debe introducir un nombre.

Si se da la situación que se muestra en la figura B.7 y hemos activado la opción de creación del fichero de errores, podremos ver que el fichero de errores tiene la información que muestra la figura B.8.Finalmente, si el establecimiento de la sesión se ha realizado de manera satisfactoria, se ejecutará el reproductor seleccionado en la pestaña de *Opciones* y comenzará a reproducir el servicio especificado.

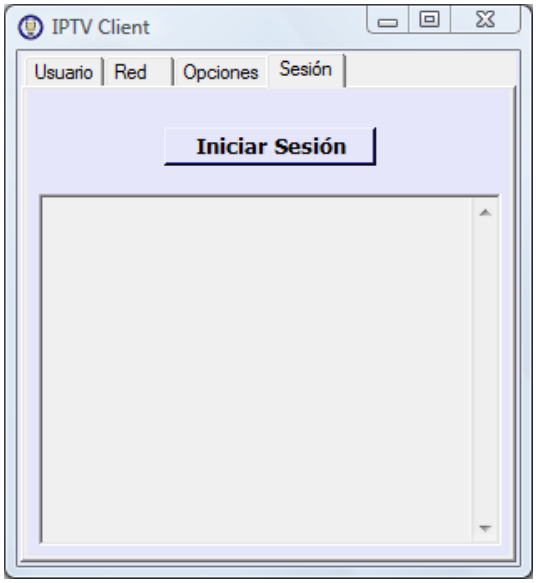

Figura B.6: Descripción de la pestaña de Sesión.

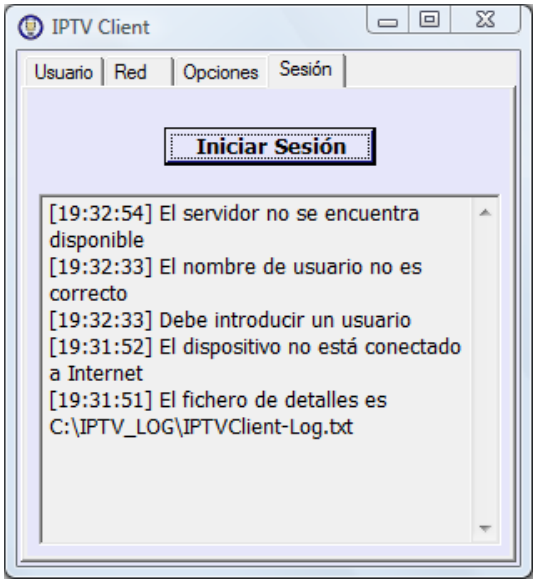

Figura B.7: Detalle de los mensajes de error mostrados.

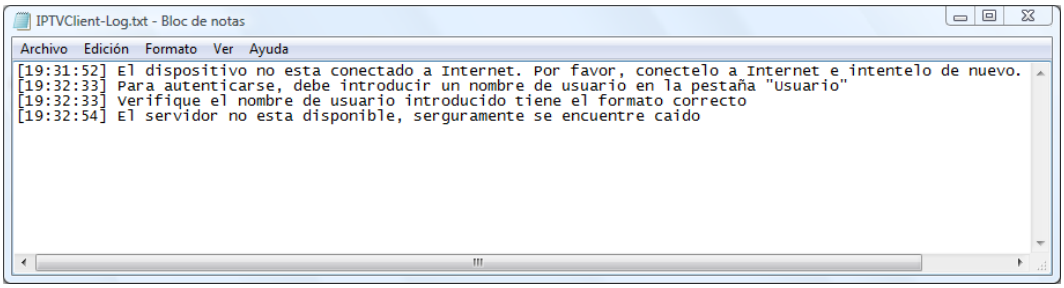

Figura B.8: Detalle del fichero de detalles.

Abreviaturas

Abreviatura	Significado
3G	Third Generation
3GPP	3rd Generation Partnership Project
4G	Fourth Generation
AAC	Advance Audio Coding
APE	Monkey's Audio
API	Application Programming Interface
AS	Application Server
ASCII	American Standard Code for Information Interchange
BGCF	Breakout Gateway Control Function
CAMEL	Customized Applications for Mobile network Enhanced Logic
CAP	CAMEL Application Part
COPS	Common Open Policy Service
CSCF	Call Session Control Functions
CSE	CAMEL Service Environment
DVB-H	Digital Video Broadcasting - Handheld
DVB-T	Digital Video Broadcasting - Terrestrial
FTP	File Transfer Protocol
FLAC	Free Lossless Audio Codec
GGSN	Gateway GPRS Support Node
GPL	General Public License
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
GUI	General User Interface
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
I-CSCF	Interrogating-CSCF
IMS	IP Multimedia Subsystem
IMS-MGW	IMS Media Gateway
IM-SSF	IP Multimedia Service Switching Function
IP	Internet Protocol
IPSec	IP Security
IPTV	IP Television
IPv4	IP version 4
IPv6	IP version 6
ISC	IMS Service Control
ISDN	Integrated Services Digital Network
ISIM	IP multimedia Services Identity Module
ISUP	ISDN User Part

Abreviatura	Significado
L1	Layer 1
LIA	Location Information Answer
LIFO	Last In First Out
LIR	Location Information Request
M3UA	SS7 MTP3-User Adaptation layer
MAP	Mobile Application Part
MBMS	Multimedia Broadcast Multicast Service
MGCF	Media Gateway Control Function
MP2	MPEG-1 Audio Layer 2
MP3	MPEG-1 Audio Layer 3
MPEG-2	Moving Pictures Experts Group 2
MRFC	Media Resource Function Controller
MRFP	Media Resource Function Processor
MTP2	Message Transfer Part level 2
MTP3	Message Transfer Part level 3
MVC	Model View Controller
NAI	Network Access Identifier
NAT	Network Address Translator
NAT-T	NAT-Traversal
NGN	Next Generation Networking
OSA	Open Service Access
OSA API	OSA Application Program Interface
OSA AS	OSA Application Server
OSA SCS	OSA Service Capability Server
P-CSCF	Proxy-CSCF
PDF	Policy Decision Function
PDP	Policy Decision Point
PPV	Pay Per View
QoS	Quality of Service
RAN	Radio Access Network
RTSP	Real-Time Streaming Protocol

Abreviatura	Significado
SA	Security Association
S-CSCF	Serving-CSCF
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SGSN	Serving GPRS Support Node
SGW	Signalling Gateway
SIP	Session Initiation Protocol
SIP AS	SIP Application Server
SLF	Subscription Locator Function
SMS	Short Message Service
SSL	Secure Sockets Layer
STUN	Session Transversal of UDP over NATs
TCP	Transmission Control Protocol
TCPMP	The Core Pocket Media Player
THIG	Topology Hiding Inter-Network Gateway
TLS	Transport Layer Security
TTA	True Audio
UDP	User Datagram Protocol
UE	User Equipment
UICC	Universal Integrated Circuit Card
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
UTF-8	8-bit Unicode Transformation Format
VOD	Video Over Demand
Windows CE	Windows Compact Edition
WAV	WAVEform Audio Format
WMV	Windows Media Video

Bibliografía

[1] Comisión del Mercado de las Telecomunicaciones, *Informe Trimestral del primer trimestre del 2009*, http://www.cmt.es/es/publicaciones/anexos/Trimestral_I_09_OK.pdf, 2009.

[2] Instituto Nacional de Estadística, *Avance del Padrón municipal a 1 de enero de 2009*, http://www.ine.es/prensa/np551.pdf, 2009.

[3] 3rd Generation Partnership Project (3GPP), *IP Multimedia Subsystem (IMS) centralized services*, http://www.3gpp.org/ftp/Specs/archive/23_series/23.292/23292-920.zip, 2008.

[4] Open IPTV Forum, *Open IPTV Forum*, http://www.openiptvforum.org/index.html, 2007.

[5] International Organization for Standardization, *Information technology – Generic coding of moving pictures and associated audio information: Systems*, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=31537, 2000.

[6] European Telecommunications Standards Institute (ETSI), *Transmission System for Handheld Terminals (DVB-H)*, http://www.dvb-h.org/PDF/DVB-HSpecification-En302304.V1.1.1.pdf, 2004.

[7] 3rd Generation Partnership Project (3GPP), *Multimedia Broadcast/Multicast Service*, http://www.3gpp.org/ftp/Specs/archive/22_series/22.146/22146-900.zip, 2008.

[8] Digital Video Broadcasting, *IP Datacast Baseline Specification, PSI/SI Guidelines for IPDC DVB-T/H Systems*, http://www.dvb.org/documents/a079.pdf, 2004.

[9] European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); AT command set for GSM Mobile Equipment*, http://www.ctiforum.com/standard/standard/etsi/0707.pdf, 1999.

[10] MediaLab, *Mobile Broadcast/Multicast Service (MBMS)*, http://www.medialab.sonera.fi/workspace/MBMSWhitePaper.pdf, 2004.

[11] International Telecommunication Union (ITU), *NGN Working definition*, http://www.itu.int/ITU-T/studygroups/com13/ngn2004/working_definition.html, 2004.

[12] 3rd Generation Partnership Project (3GPP), *Service requirements for the Internet Protocol (IP) multimedia core network subsystem (IMS)*, http://www.3gpp.org/ftp/Specs/archive/22_series/22.228/22228-500.zip, 2001.

[13] 3rd Generation Partnership Project (3GPP), *Service principles*, http://www.3gpp.org/ftp/Specs/html-info/22101.htm, 1999.

[14] International Telecommunication Union (ITU), *Asymmetric digital subscriber line (ADSL) transceivers - Extended bandwidth ADSL2 (ADSL2plus)*, http://www.itu.int/rec/T-REC-G.992.5-200901-P/en, 2009.

[15] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), June 2002, Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[16] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401 (Proposed Standard), Nov. 1998, Obsoleted by RFC 4301, updated by RFC 3168.

[17] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617 (Draft Standard), June 1999.

[18] 3rd Generation Partnership Project (3GPP), *Technical Specification Group Core Network and Terminals; Customised Applications for Mobile network Enhanced Logic (CAMEL) Phase 4;CAMEL Application Part (CAP) specification*, http://www.3gpp.org/ftp/Specs/archive/29_series/29.078/29078-800.zip, 2008.

[19] H. Schulzrinne, "The tel URI for Telephone Numbers," RFC 3966 (Proposed Standard), Dec. 2004, Updated by RFC 5341.

[20] B. Aboba and M. Beadles, "The Network Access Identifier," RFC 2486 (Proposed Standard), Jan. 1999, Obsoleted by RFC 4282.

[21] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321 (Informational), Apr. 1992.

[22] D. Eastlake 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174 (Informational), Sept. 2001, Updated by RFC 4634.

[23] Gonzalo Camarillo y Miguel A. García-Martín, *The 3G IP Multimedia Subsystem (IMS). Merging the Internet and the Cellular World*, Wiley, second edition edition, 2006.

[24] Hisham Khartabil y Aki Niemi Miikka Poikselkä, Georg Mayer, *The IMS IP Multimedia Concepts and Services*, Wiley, second edition edition, 2006.

[25] V. Cerf, Y. Dalal, and C. Sunshine, "Specification of Internet Transmission Control Program," RFC 675, Dec. 1974.

[26] J. Postel, "User Datagram Protocol," RFC 768 (Standard), Aug. 1980.

[27] Antisip, *GNU osip*, http://www.gnu.org/software/osip/, 2000.

[28] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566 (Proposed Standard), July 2006.

[29] Antisip, *eXosip*, http://savannah.nongnu.org/projects/exosip/, 2003.

[30] Nokia Research Center, *Sofia-SIP*, http://research.nokia.com/research/projects/sofia-sip/index.html, 2006.

[31] J. Postel, "Internet Protocol," RFC 791 (Standard), Sept. 1981, Updated by RFC 1349.

[32] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), Dec. 1998, Updated by RFC 5095.

[33] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389 (Proposed Standard), Oct. 2008.

[34] Erik Eliasson, *Minisip*, http://www.minisip.org/index.html, 2004.

[35] Ivar Lumi, *LumiSoft.Net*, http://www.lumisoft.ee/lsWWW/download/downloads/Net, 2009.

[36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), June 1999, Updated by RFC 2817.

[37] J. Postel and J. Reynolds, "File Transfer Protocol," RFC 959 (Standard), Oct. 1985, Updated by RFCs 2228, 2640, 2773, 3659.

[38] Microsoft Windows, *Windows Media Player 10 Mobile*, http://www.microsoft.com/windowsmobile/es-es/help/more/windows-media-player.mspx, 2009.

[39] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2326 (Proposed Standard), Apr. 1998.

[40] 3rd Generation Partnership Project (3GPP), *Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP)*, http://www.3gpp.org/ftp/Specs/archive/26_series/26.244/26244-810.zip, 2009.

[41] Beta Player, *The Core Pocket Media Player*, http://picard.exceed.hu/tcpmp/test/, 2009.

[42] VideoLan, *VideoLan Client*, http://www.videolan.org/vlc/, 2005.

[43] Steve Burbeck, *How to use Model-View-Controller (MVC)*, http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html, 1992.

[44] VideoLAN, *VideoLAN Wiki - WinCECompile*, http://wiki.videolan.org/WinCECompile, 2008.